Chapter 1

Utilities

This section contains reference pages for the INTERIX Commands and Utilities, Release 2.2. Each reference page contains these sections:

NAME The name of the utility.

SYNOPSIS
How to type a command line using the utility. Optional flags and arguments are shown in brackets.

DESCRIPTION
What the utility does, the effect of each option flag, and what happens with each argument.

EXAMPLE
An example of a command line using the utility. This section is optional.

DIAGNOSTICS
The possible return values of the utility or a description of error messages. This section is optional.

NOTES Additional information that may be helpful. This section is optional.

SEE ALSO
Pointers to related utilities or system interfaces. This section is optional.

Some reference pages have additional sections unique to the utility. For example, the *awk* reference page has sections describing the *awk* language. The *lex* reference page has a second reference page, *lexdoc*, which describes *lex* input files.

The INTERIX Commands and Utilities is a conforming IEEE 1003.2-1992 POSIX Shell and Utilities implementation. (This is entirely equivalent to ISO/IEC 9945-2:1993.) This means that the ultimate reference for how a utility behaves is found in the standard. The reference pages here are simple quick references, and do not go to the depths of description that a *de jure* standard requires. The POSIX.2 Conformance Document contains tersely-worded descriptions of how individual utilities implement areas of the standard that are labelled *implementation defined*.

Much of the text of these reference pages was derived from the reference pages shipped with the 4.4BSD distribution and as such bear a University of California copyright. (Please see the *Copyrights* document for complete information.)

NAME

appres – list application resource database

SYNOPSIS

`appres [`*`class`* `[`*`instance`*`]] [-1] [`*`toolkitoptions`*`]`

DESCRIPTION

The *appres* program prints the resources seen by an application (or subhierarchy of an application) with the specified *class* and *instance* names. It can be used to determine which resources a particular program will load. For example,

```
% appres XTerm
```

will list the resources that any *xterm* program will load. If no application class is specified, the class `-AppResTest-` is used.

To match a particular instance name, specify an instance name explicitly after the class name, or use the normal Xt toolkit option. For example,

```
% appres XTerm myxterm
```

or

```
% appres XTerm -name myxterm
```

To list resources that match a subhierarchy of an application, specify hierarchical class and instance names. The number of class and instance components must be equal, and the instance name should not be specified with a toolkit option. For example,

```
% appres Xman.TopLevelShell.Form xman.topBox.form
```

will list the resources of widgets of *xman* topBox hierarchy. To list just the resources matching a specific level in the hierarcy, use the –1 option. For example,

```
% appres XTerm.VT100 xterm.vt100 -1
```

will list the resources matching the *xterm* vt100 widget.

SEE ALSO

X, *xrdb*, *listres*

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

a2p – Awk to Perl translator

SYNOPSIS

`a2p [`*`options`*`] `*`filename`*

DESCRIPTION

A2p takes an *awk* script specified on the command line (or from standard input) and produces a comparable *perl* script on the standard output.

Options

Options include:

–D *number*

Set debugging flags.

–F*character*

Tell *a2p* that this *awk* script is always invoked with this **–F** switch.

–n*fieldlist*

Specify the names of the input fields if input does not have to be split into an array. If you were translating an *awk* script that processes the password file, you might say:

```
a2p -7 -nlogin.password.uid.gid.gcos.shell.home
```

Any delimiter can be used to separate the field names.

–*number*

causes *a2p* to assume that input will always have that many fields.

Considerations

A2p cannot do as good a job translating as a human would, but it usually does pretty well. There are some areas where you may want to examine the perl script produced and tweak it some. Here are some of them, in no particular order.

There is an *awk* idiom of putting *int*( ) around a string expression to force numeric interpretation, even though the argument is always integer anyway. This is generally unneeded in *perl*, but *a2p* can't tell if the argument is always going to be integer, so it leaves it in. You may wish to remove it.

Perl differentiates numeric comparison from string comparison. *Awk* has one operator for both that decides at run time which comparison to do. *A2p* does not try to do a complete job of *awk* emulation at this point. Instead it guesses which one you want. It's almost always right, but it can be spoofed. All such guesses are marked with the comment ''#???''. You should go through and check them. You might want to run at least once with the **–w** switch to *perl*, which will warn you if you use **==** where you should have used **eq**.

Perl does not attempt to emulate the behavior of *awk* in which nonexistent array elements spring into existence simply by being referenced. If somehow you are relying on this mechanism to create null entries for a subsequent **for** ... **in**, they won't be there in *perl*.

If *a2p* makes a split line that assigns to a list of variables that looks like (Fld1, Fld2, Fld3...) you may want to rerun *a2p* using the **–n** option mentioned above. This will let you name the fields throughout the script. If it splits to an array instead, the script is probably referring to the number of fields somewhere.

The **exit** statement in *awk* doesn't necessarily exit; it goes to the END block if there is one. *Awk* scripts that do contortions within the END block to bypass the block under such circumstances can be simplified by removing the conditional in the END block and just exiting directly from the *perl* script.

Perl has two kinds of array, numerically-indexed and associative. *Awk* arrays are usually translated to associative arrays, but if you happen to know that the index is always going to be numeric you could change the {...} to [...]. Iteration over an associative array is done using the *keys*( ) function, but iteration over a numeric array is *not*. You might need to modify any loop that is iterating over the array in question.

Awk starts by assuming *OFMT* has the value %.6g. *Perl* starts by assuming its equivalent, *$#*, to have the value %.20g. You'll want to set *$#* explicitly if you use the default value of *OFMT*.

Near the top of the line loop will be the split operation that is implicit in the *awk* script. There are times when you can move this down past some conditionals that test the entire record so that the split is not done as often.

For aesthetic reasons you may wish to change the array base $[from 1 back to *perl*'s default of 0, but remember to change all array subscripts *and* all *substr*( ) and *index*( ) operations to match.

Cute comments that say "# Here is a workaround because awk is dumb" are passed through unmodified.

Awk scripts are often embedded in a shell script that pipes stuff into and out of *awk*. Often the shell script wrapper can be incorporated into the *perl* script, since *perl* can start up pipes into and out of itself, and can do other things that *awk* can't do by itself.

Scripts that refer to the special variables *RSTART* and *RLENGTH* can often be simplified by referring to the variables *$`*, *$&* and *$'*, as long as they are within the scope of the pattern match that sets them.

The produced *perl* script may have subroutines defined to deal with *awk*'s semantics regarding **getline** and **print**. Since *a2p* usually picks correctness over efficiency. it is almost always possible to rewrite such code to be more efficient by discarding the semantic sugar.

For efficiency, you may wish to remove the keyword from any return statement that is the last statement executed in a subroutine. *A2p* catches the most common case, but doesn't analyze embedded blocks for subtler cases.

ARGV[0] translates to $ARGV0, but ARGV[n] translates to $ARGV[$n]. A loop that tries to iterate over ARGV[0] won't find it.

ENVIRONMENT VARIABLES

A2p uses no environment variables.

AUTHOR

Larry Wall <lwall@jpl-devvax.Jpl.Nasa.Gov>

SEE ALSO

perl (the *perl* compiler/interpreter), *s2p* (*sed* to *perl* translator)

DIAGNOSTICS

On success, *a2p* exits with status 0; otherwise, it exist with a status >0.

BUGS

It would be possible to emulate *awk*'s behavior in selecting string versus numeric operations at run time by inspection of the operands, but it would be gross and inefficient. Besides, *a2p* almost always guesses right.

Storage for the *awk* syntax tree is currently static, and can run out.

The version of *a2p* distributed with 5.003 has known problems with multiple concatenations and redirection.

Concatenations must be in pairs:

```
var="x" "y" "z"
```

must be done as:

```
var=( "x" "y" ) "z"
```

to translate; otherwise *a2p* prints a message about syntax errors. (This bug did not exist in *perl* 4.x versions of *a2p*.)

Redirection such as:

```
{ print("foo") > "bar" }
```

will not be translated correctly. You may want to comment out redirection lines and rewrite them in *perl* directly.

NAME

ar – create and maintain library archives

SYNOPSIS

```
ar –d [–v] archive file ...
ar –m [–abilv] [posname] archive file ...
ar –p [–v] archive file ...
ar –q [ –clv] archive file ...
ar –r [–cuv] archive file ...
ar –t [–v] archive [file ...]
ar –x [–v] archive [file ...]
```

DESCRIPTION

Ar creates and maintains groups of *file*s combined into an *archive*. After an archive has been created, *ar* can be used to modify the contents by adding new files and by deleting, extracting, or replacing existing files.

The most common use of *ar* is to combine object files into a library archive. If all of the specified *file*s are valid object files, *ar* automatically creates a library. If some of the specified *file*s are not valid object files (for example, are text files), the library may not be valid.

The first option given to *ar* specifies the type of action; the remaining options modify that action.

- To *create* an archive, use **–r**.
- To *delete* an archive member, use **–d**.
- To *replace* existing archive members or to *add* a member to the archive, use **–r**.
- To *print* the contents of an archive, use **–p**.
- To provide a *table of contents* for the archive, use **–t**.
- To *extract* members from the archive, use **–x**.

The *file* can be a pathname if the file is being added to the archive from the system (whether as a replacement or a new file). If the file is already stored in the archive (for example, when moving, deleting, or extracting), *file* should be the last component of the member's pathname.

Ar takes the following options:

–a Add new files to the archive after the member file named by *posname*.

–b add new files to the archive before the member file named by *posname*.

–c Suppress diagnostic message; normally, *ar* writes a message to standard error when *archive* is created. This option is only useful with **–c** and **–r**.

–C When extracting files, protect existing files in the file system from being overwritten by members with similar names. This is particularly useful with the **–T** option, to prevent truncated filenames from being overwritten by files with the same prefix.

–d Delete one or more *file*s from the *archive*.

–i Insert the new files to the archive before the member file named by *posname* (identical to **–b**).

–l Work locally, by placing the temporary file in the current directory, rather than in the directory specified by *TMPDIR*.

–m Move the named *file*s within the *archive*; unless the **–a**, **–b**, or **–i** options are given, the files are moved to the end of the archive.

–p Print the contents of the *file*s from the *archive* to standard output. If no files are specified, print the entire contents of the archive. (For a list of the archive's contents, use **–t**.)

–q Quickly append the named *file*s to the end of the *archive*, without checking to see if the files are already members of the archive. This option is useful when building a large archive a piece at a time.

–r Replace or add the *file*s to the *archive*. If *archive* doesn't exist, create it. Files that are added (rather than replacing existing members) are added to the end of the archive.

–t Write the *archive*'s table of contents to standard output. If no *file*s are specified, all members are written; otherwise, only the specified *file*s are written.

–T Truncate file names if they are too long for the file system. Normally, *ar* refuses to extract files if the names are too long.

–u Update files; when this option is used with **–r**, *ar* only replaces files if the modification time on the file is newer than the modification time of the archive member.

–v Provide verbose output.

–x Extract the named *file*s from the *archive*. The archive itself is not changed. If the *file*'s name is too long for the current file system, it is not extracted . If no *file*s are specified, all files are extracted.

DIAGNOSTICS

Ar can return one of the following exit values:

0 Success.

>0 An error occurred.

ENVIRONMENT VARIABLES

The *ar* utility makes use of the following environment variables, if set:

TMPDIR

Use this directory for temporary files, rather than the default.

SEE ALSO

c89

NAME

as – the portable GNU assembler.

SYNOPSIS

```
as [-DfKLMRvw] [-a[dhlns] [=file]] [--defsym SYM=VAL]
   [-I path] [-M|--mri] [-o objfile]
   [--|files...]

i960-onlyoptions:
   [-ACA|-ACA_A|-ACB|-ACC|-AKA|-AKB|-AKC|-AMC]
   [-b] [-no-relax]

m680x0-onlyoptions:
   [-l] [-mc68000|-mc68010|-mc68020]
```

DESCRIPTION

GNU *as* is really a family of assemblers. If you use (or have used) the GNU assembler on one architecture, you should find a fairly similar environment when you use it on another architecture. Each version has much in common with the others, including object file formats, most assembler directives (often called *pseudo-ops)* and assembler syntax.

For information on the syntax and pseudo-ops used by GNU *as*, see *as* entry in *info* (or the manual *Using as: The GNU Assembler*).

as is primarily intended to assemble the output of the GNU C compiler *gcc* for use by the linker *ld*. Nevertheless, we've tried to make *as* assemble correctly everything that the native assembler would. This doesn't mean *as* always uses the same syntax as another assembler for the same architecture; for example, we know of several incompatible versions of 680x0 assembly language syntax.

Each time you run *as* it assembles exactly one source program. The source program is made up of one or more files. (The standard input is also a file.)

If *as* is given no file names it attempts to read one input file from the *as* standard input, which is normally your terminal. You may have to type **Control-D** to tell *as* there is no more program to assemble. Use **––** if you need to explicitly name the standard input file in your command line.

as may write warnings and error messages to the standard error file (usually your terminal). This should not happen when *as* is run automatically by a compiler. Warnings report an assumption made so that *as* could keep assembling a flawed program; errors report a grave problem that stops the assembly.

OPTIONS

–a *suboptions*

Turn on assembly listings. There are various *suboptions*. **d** omits debugging directives. **h** includes the high level source code; this is only available if the source file can be found, and the code was compiled with **–g**. **l** includes an assembly listing. **n** omits forms processing. **s** includes a symbol listing. **=** *file* sets the listing file name; this must be the last suboption. The default suboptions are **hls**.

–D This option is accepted only for script compatibility with calls to other assemblers; it has no effect on *as*.

—defsym *SYM=VALUE*
Define the symbol *SYM* to be *VALUE* before assembling the input file. *VALUE* must be an integer constant. As in C, a leading 0x indicates a hexadecimal value, and a leading 0 indicates an octal value.

–f Fast—skip preprocessing (assume source is compiler output).

–I *path* Add *path* to the search list for **.include** directives.

–K Issue warnings when difference tables altered for long displacements.

–L Keep (in symbol table) local symbols, starting with **L**

–M, **—mri** Assemble in MRI compatibility mode.

–o *objfile* Name the object-file output from *as*

–R Fold data section into text section.

–v Announce *as* version

–W Suppress warning messages

––*files* . . .
Source files to assemble, or standard input ((**––**)

–A *var* *(When configured for Intel 960.)*
Specify which variant of the 960 architecture is the target.

–b *(When configured for Intel 960.)*
Add code to collect statistics about branches taken.

–no-relax
(When configured for Intel 960.)
Do not alter compare-and-branch instructions for long displacements; error if necessary.

–l *(When configured for Motorola 68000).*
Shorten references to undefined symbols, to one word instead of two.

–mc68000 | **–mc68010** | **–mc68029**
(When configured for Motorola 68000).
Specify what processor in the 68000 family is the target (default 68020)

Options may be in any order, and may be before, after, or between file names. The order of file names is significant.

The option **––** (two hyphens) by itself names the standard input file explicitly, as one of the files for *as* to assemble.

Except for **––** any command line argument that begins with a hyphen (**–**) is an option. Each option changes the behavior of *as*. No option changes the way another option works. An option is a **–** followed by one or more letters; the case of the letter is important. All options are optional.

The **–o** option expects exactly one file name to follow. The file name may either immediately follow the option's letter (compatible with older assemblers) or it may be the next command argument (GNU standard).

These two command lines are equivalent:

```
as  -o  my-object-file.o  mumble.s
as  -omy-object-file.o  mumble.s
```

COPYING

Copyright (c) 1991, 1992 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

SEE ALSO

gcc, *ld*

as entry in *info*; *Using as: The GNU Assembler*.

NAME

at, atq, atrm – run commands at a later time

SYNOPSIS

```
at [-mVv] [-f file] [-q queuename] -t time
at [-mVv] [-f file] [-q queuename] timespec
at [–V] –c at_job_id ...
at [–V] –l -q queuename
at [–V] –l [at_job_id ...]
at [–V] –r at_job_id ...

atq [–Vv] [–q queuename]
atrm [–V] at_job_id ...
```

DESCRIPTION

The *at* utility runs commands at a later time. Commands are read from standard input and grouped together into an *at-job*; the at-job has a numeric identifier, the *at_job_id*, which can be used with *at* to change the job later.

When you submit an at-job using one of the first two forms, *at* writes the scheduled time and the *at_job_id* to standard output. The time can be given as an argument to the **–t** option or as a *timespec*; the format of the *timespec* is given later.

The third form of the command (with **–r**) is used to remove the specified job from the schedule.

The fourth and fifth forms of the command (with **–l**, the letter ell) list all queued jobs.

At takes the following options:

–c Copy job file(s) specified by *at_job_id* to standard output).

–f *file* Take the list of commands to be run from *file* instead of standard output.

–L List your jobs in a long format, with column titles.

–l List your jobs. If no *at_job_id* is specified, list all jobs for the current user; otherwise list information about the specified job or jobs. If **–q** is also given, list jobs in the specified queue. (On other systems, the **–l** option lists *all* jobs; on INTERIX, the option lists only your own jobs, since each user's *at* queue is administered by a separate *cron* daemon.)

–m Send user notification that the job was run, by electronic mail; the mail includes the standard output and standard error of the job (unless it was redirected elsewhere). If the job produces standard output or standard error, it is mailed to the user whether or not the **–m** option is given, unless it is redirected elsewhere.

–q *queuename*
Schedule the job in the specified queue. The *queuename* can be either **a** (the default for jobs scheduled with *at*) or **b** (for batch jobs scheduled with *batch*).

–r Remove the specified job(s) from the schedule.

–t *time* Run the command at the specified *time*; *time* has the same format as the time argument to the *touch* command.

–V Print version number of program.

–v Validate time specifications.

When the job runs, it runs in a separate shell (**$OPENNT_ROOT/bin/sh**) and the environment variables, current working directory, and other execution-time attributes are the same as when the *at* command was given.

You can force the job to run in a different shell by explicitly stating the name of the shell. For example, invoking *at* in the Korn shell to run *myscript* using the C shell:

```
at now <<eof
csh -c myscript
eof
```

Since the shell inherits your *PATH*, there's no need to specify a full pathname (unless the command is not found in your *PATH*).

The command *atq* is equivalent to **at –L**; the command *atrm* is equivalent to **at –r**.

Rather than using the **at.deny** and **at.allow** files found on some systems, this implementation uses *cron* to run a command named *atrun*. See the "NOTES" below.

Mail to the user is sent using the *mailx* utility. You can change the mail delivery agent by changing the mail delivery agent for *mailx* (see the *mailx* reference page).

The Time Specification

A time specification is a *time*, an optional *timezone*, and an optional *date*:

time **[***timezone***] [***date***]**

The *time* can be given numerically, or as one of the words **midnight**, **noon**, **teatime** (4 P.M.), or **now**.

You can specify the time numerically using a one, two, or four-digit number; a one or two digit number specifies the hours.

h
hh
hhmm
hh:*mm*

Unless you put **AM** or **PM** after the time, it's assumed to be a 24-hour clock. The time **11** is 11:00 AM; the times **11 PM**, **11:00 PM**, **23**, **2300**, and **23:00** are all 11:00 PM.

After the time, you can specify a *timezone* name. The *timezone* string is case insensitive. If no timezone is specified, it's assumed to be in the timezone specified by the *TZ* environment variable. The only value currently accepted for the timezone is the string **utc** (for Co-ordinated Universal Time).

A *date* is a month name followed by a day number (optionally followed by a comma and a year number), or a day of the week, or one of the words **today** or **tomorrow**. You can get a list of valid month names with the command **locale -ck mon abmon**. You can get a list of valid weekday names with the command **locale -ck day abday**.

A time or time and date can also have an *increment* after it, so you can say (for example) **noon+5 minutes** or **midnight Aug 5 + 1 hour**. An increment is always **+**, a number, and one of the units **minute**, **minutes**, **hour**, **hours**, **day**, **days**, **week**, **weeks**, **month**, **months**, **year**, or **years**. As a special case, you can use the word **next** instead of **+1**:

```
at noon next week
```

specifies noon in seven days.

DIAGNOSTICS

After successfully queuing, listing, or removing a job or jobs, *at* returns 0. If an error occurred, *at* returns >0.

ENVIRONMENT VARIABLES

The *at* command makes use of the following environment variables, if set.

SHELL Specifies the name of the command interpreter used to run the command. Defaults to *sh*.

TZ The value of the timezone environment variable determines the timezone used.

NOTES

In this implementation, the actual interface between *at* and the system scheduler *cron* is a program called *atrun*, which by default runs every 10 minutes. Once a user's *cron* service is enabled (see *cronadmin*), *atrun* begins sending mail about jobs. This may be distracting. While it's not possible to disable *atrun* totally without turning off *cron*, the *atrun* job can be rescheduled in the **crontab** file (using **crontab –e**). Any command line containing the string "atrun" overrides the default 10-minute schedule. For example, adding this command line to the **crontab** file will cause *atrun* to be invoked only on New Year's day, at one minute after midnight:

```
01 01 01 01 * atrun
```

FILES

The *at* and *atrun* commands make use of the following files:

$OPENNT_ROOT/usr/lib/cron/user/*userid***/at/jobs/***** The *at* job directory, containing jobs to be run.

SEE ALSO

batch, *cronadmin*, *crontab*, *locale*, *touch*

NAME

atrun – run jobs queued for later execution

SYNOPSIS

`atrun [-l load_avg] [-d]`

DESCRIPTION

Atrun runs jobs queued by *at*. If there is no call to *atrun* in the user's **crontab** file, the *cron1* command in INTERIX calls *atrun* every ten minutes.

At every invocation, every job in lowercase queues whose starting time has passed is started. A maximum of one batch jobs (denoted by uppercase queues) are started each time *atrun* is invoked.

The *atrun* command takes the following options.

–l *load_avg* Specifies a limiting load factor, over which batch jobs should not be run, instead of the compiled-in value of 1.5. Because there is no concept of load factor in Windows NT, this option has no effect.

–d Debug; print error messages to standard error instead of using *syslog*( ).

WARNINGS

For *atrun* to work, you have to start up a *cron* daemon.

NOTES

The original author is Thomas Koenig <ig25@rz.uni-karlsruhe.de>.

FILES

$OPENNT_ROOT/usr/lib/cron/user/*userid***/at/jobs/***

Directory containing job files.

SEE ALSO

at, *cron*, *crontab*

syslog()

crontab(5)

BUGS

The functionality of *atrun* should be merged into *cron*.

NAME

awk – pattern-directed scanning and processing language

SYNOPSIS

awk **[***-F fs***] [***-v var=value***] [***'prog'* **|** -f *progfile***] [***file ...***]**

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified literally in *prog* or in one or more files specified as **–f** *progfile*. With each pattern there can be an associated action that will be performed when a line of a *file* matches the pattern. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. The file name – means the standard input. Any *file* of the form *var=value* is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename. The option **–v** followed by *var=value* is an assignment to be done before *prog* is executed; any number of **–v** options may be present. The **–F** *fs* option defines the input field separator to be the regular expression *fs*.

An input line is normally made up of fields separated by white space. (This default can be changed by using the FS built-in variable or the **–F** *fs* option.) The fields are denoted **$1**, **$2**, ..., while $0 refers to the entire line.

A pattern-action statement has the form

pattern { *action* }

A missing { action } means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

An action is a sequence of statements. A statement can be one of the following, listed in approximately lexical order:

Begin comment; continues to end-of-line.

break Leave innermost enclosing **for**, **do**, or **while** loop.

continue
Skip to next iteration of innermost enclosing **for**, **do**, or **while** loop.

{ [*statement* ...] **}**

delete *array*[*expression*]
Delete an array element.

do *statement* **while(** *expression* **)**
Execute *statement* and repeat so long as *expression* is true.

exit [*expression*]
Move immediately to the END action; if in the END action, exit immediately. Exit status is *expression*.

expression
Commonly *var* **=** *expression*

for (*expression1* **;** *expression2* **;** *expression3* **)** *statement*
Conditional loop, executing *statement* so long as *expression2* is true. This is equivalent to *expression1***; while (***expression2***) {** *statement***;** *expression3* **}**

for (*var* **in** *array* **)** *statement*
Loop iterating over array contents.

if (*expression* **)** *statement1* [**else** *statement2*]
Conditional statement: execute *statement1* if *expression* is true; otherwise, execute *statement2*.

next
Skip remaining patterns on this input line and start the next input line.

print [*expression-list*] [> *expression*]
Print output.

printf *format* [**,** *expression-list*] [> *expression*]
Print formatted output.

return [*expression*]
Return from function.

while(*expression* **)** *statement*
Conditional loop: so long as *expression* is true, execute *statement*.

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for $0 . String constants are quoted " "; the usual C escapes are recognized within strings. Expressions take on string or numeric values as appropriate, and are built using the operators + - * / % ^ (exponentiation), and concatenation (indicated by a blank). The operators ! ++ -- += -= *= /= %= ^= > >= < <= == != ?: are also available in expressions. Variables may be scalars, array elements (denoted *x*[*i*]*)* or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. Multiple subscripts such as **[i,j,k]** are permitted; the constituents are concatenated, separated by the value of **SUBSEP**.

The **print** statement prints its arguments, separated by the current output field separator, and terminated by the output record separator. By default it prints to standard output, but it prints to a file if >*file* or >>*file* is present or on a pipe if |*cmd* is present). *file* and *cmd* may be literal names or parenthesized expressions; identical string values in different statements denote the same open file. The **printf** statement formats its expression list according to the format similar to *printf*( ). The built-in function **close(***expr***)** closes the file or pipe *expr*.

The mathematical functions *exp*( ), *log*( ), *sqrt*( ), *sin*( ), *cos*( ), and *atan2*( ) are built in. Other built-in functions include:

length the length of its argument taken as a string, or of $0 if no argument.

rand random number on (0,1)

srand sets seed for *rand*( ) and returns the previous seed.

int truncates to an integer value

substr(*s***,** *m***,** *n***)**
the *n*-character substring of *s* that begins at position *m* counted from 1.

index(*s***,** *t***)**
the position in *s* where the string *t* occurs, or 0 if it does not.

match(*s***,** *r***)**
the position in *s* where the regular expression *r* occurs, or 0 if it does not. The variables **RSTART** and **RLENGTH** are set to the position and length of the matched string.

split(*s***,** *a***,** *fs***)**
splits the string *s* into array elements *a*[1], *a*[2], ..., *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator **FS** if *fs* is not given.

sub(*r***,** *t***, s)** substitutes *t* for the first occurrence of the regular expression *r* in the string *s*. If *s* is not given, $0 is used.

gsub same as *sub*( ) except that all occurrences of the regular expression are replaced; *sub*( ) and *gsub*( ) return the number of replacements.

sprintf(*fmt***,** *expr***, ...)**
the string resulting from formatting *expr*... according to the *printf*( ) function format *fmt*.

system(*cmd***)**
executes *cmd* and returns its exit status

The "function" **getline** sets $0 to the next input record from the current input file; **getline <** *file* sets $0 to the next record from *file*. **getline x** sets variable *x* instead. Finally, *cmd* **|** **getline** pipes the output of *cmd* into **getline**; each call of **getline** returns the next line of output from *cmd*. In all cases, **getline** returns 1 for a successful input, 0 for end of file, and –1 for an error.

Patterns are arbitrary Boolean combinations (with **! | | &&**) of regular expressions and relational expressions. Regular expressions are as in POSIX.2 Extended Regular Expressions (ERE). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions, using the operators **~** and **!~**. */re/* is a constant regular expression; any string (constant or variable) may be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines from an occurrence of the first pattern though an occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression
expression **in** *array-name*
(*expr***,***expr*,...**) in** *array-name*

where a *relop* is any of the six relational operators in C, and a *matchop* is either **~** (matches) or **!~** (does not match). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns **BEGIN** and **END** may be used to capture control before the first input line is read and after the last. **BEGIN** and **END** do not combine with other patterns.

Variable names with special meanings:

Variable	Meaning
ARGC	argument count, assignable
ARGV	argument array, assignable; non-null members are taken as filenames
ENVIRON	array of environment variables; subscripts are names.
FILENAME	the name of the current input file
FNR	ordinal number of the current record in the current file
FS	regular expression used to separate fields; also set by option **–F** *fs*
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default "%.6g")
OFS	output field separator (default blank)
ORS	output record separator (default newline)
RS	input record separator (default newline)
SUBSEP	separates multiple subscripts (default 034)

Functions may be defined (at the position of a pattern-action statement) thus:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar and by reference if array name; functions may be called recursively. Parameters are local to the function; all other variables are global. Thus local variables may be created by providing excess parameters in the function definition.

EXAMPLES

Print lines longer than 72 characters.

```
length > 72
```

Print first two fields in opposite order.

```
{ print $2, $1 }
```

Same, with input fields separated by comma and / or blanks and tabs.

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

Add up first column, print sum and average.

```
       { s += $1 }
END    { print "sum is", s, " average is", s/NR }
```

NOTES

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate "" to it.

SEE ALSO

sed

NAME

banner – print large banner on printer

SYNOPSIS

`banner [-w n] message ...`

DESCRIPTION

Banner prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If **–w** is given, the output is scrunched down from a width of 132 to *n*, suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is great enough that you may want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

BUGS

Several ASCII characters are not defined, notably <, >, [,], \, ^, _, {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The **–w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

AUTHOR

Mark Horton

NAME

basename – return filename portion of pathname

SYNOPSIS

`basename string [`*`suffix`*`]`

DESCRIPTION

Basename deletes any prefix ending with the last slash / character present in *string*, and a *suffix*, if given. The resulting filename is written to the standard output. If *string* ends in the slash character, /, or is the same as the *suffix* argument, a newline is output. A non-existent suffix is ignored.

The *basename* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

sh

NAME

batch – run commands when system load permits

SYNOPSIS

```
batch
```

DESCRIPTION

The *batch* command is equivalent to:

```
at -q b -m now
```

Unlike the default behavior of *at*, *batch* lets you submit jobs for immediate execution subject to unspecified scheduling delays. It can be thought of as the polite way to schedule jobs.

When *batch* submits a job, it writes to standard output the *at_job_id* of the job and the current date and time. The *at_job_id* can be used to cancel the job (see *at*).

Because INTERIX does not yet support load averaging or the job priorities through *nice*, the queues do not have different priorities.

DIAGNOSTICS

When the command has been successfully scheduled, *batch* exits with a 0 status. Otherwise, it exits with >0.

SEE ALSO

at, *atrun*, *crontab*

NAME

bc – An arbitrary precision calculator language

SYNOPSIS

`bc [-lws] [file ...]`

DESCRIPTION

bc is a language that supports arbitrary precision numbers with interactive execution of statements. There are some similarities in the syntax to the C programming language. A standard math library is available by command line option. If requested, the math library is defined before processing any files. *bc* starts by processing code from all the files listed on the command line in the order listed. After all files have been processed, *bc* reads from the standard input. All code is executed as it is read. (If a file contains a command to halt the processor, *bc* will never read from the standard input.)

This version of *bc* contains several extensions beyond traditional *bc* implementations and the POSIX draft standard. Command line options can cause these extensions to print a warning or to be rejected. This document describes the language accepted by this processor. Extensions will be identified as such.

bc takes the following command line options:

–l Define the standard math library.

–w Give warnings for extensions to POSIX *bc*.

–s Process exactly the POSIX *bc* language.

NUMBERS

The most basic element in *bc* is the number. Numbers are arbitrary precision numbers. This precision is both in the integer part and the fractional part. All numbers are represented internally in decimal and all computation is done in decimal. (This version truncates results from divide and multiply operations.) There are two attributes of numbers, the length and the scale. The length is the total number of significant decimal digits in a number and the scale is the total number of decimal digits after the decimal point. For example:

.000001 has a length of 6 and scale of 6.
1935.000 has a length of 7 and a scale of 3.

VARIABLES

Numbers are stored in two types of variables, simple variables and arrays. Both simple variables and array variables are named. Names begin with a letter followed by any number of letters, digits and underscores. All letters must be lower case. (Full alpha-numeric names are an extension. In POSIX *bc* all names are a single lower case letter.) The type of variable is clear by the context because all array variable names will be followed by brackets ([]).

There are four special variables, **scale**, **ibase**, **obase**, and **last**.

scale defines how some operations use digits after the decimal point. The default value of **scale** is 0.

ibase defines the conversion base for input numbers. The default is base 10.

obase defines the conversion base for input numbers. The default is base 10.

last (an extension) is a variable that has the value of the last printed number. In some versions of *bc*, this is variable has the name . (dot).

All of these variables may have values assigned to them as well as used in expressions. These variables will be discussed in further detail where appropriate.

COMMENTS

Comments in *bc* start with the characters **/*** and end with the characters ***/**. Comments may start anywhere and appear as a single space in the input. (This causes comments to delimit other input items. For example, a comment can not be found in the middle of a variable name.) Comments include any newlines (end of line) between the start and the end of the comment.

EXPRESSIONS

The numbers are manipulated by expressions and statements. Since the language was designed to be interactive, statements and expressions are executed as soon as possible. There is no "main" program. Instead, code is executed as it is encountered. (Functions, discussed in detail later, are defined when encountered.)

A simple expression is just a constant. *bc* converts constants into internal decimal numbers using the current input base, specified by the variable **ibase**. (There is an exception in functions.) The legal values of **ibase** are 2 through 16 (F). Assigning a value outside this range to **ibase** will result in a value of 2 or 16. Input numbers may contain the characters 0-9 and A-F. (Note: They must be capitals. Lower case letters are variable names.) Single digit numbers always have the value of the digit regardless of the value of **ibase**. (i.e. A = 10.) For multi-digit numbers, *bc* changes all input digits greater or equal to **ibase** to the value of **ibase**–1. This makes the number FFF always be the largest 3 digit number of the input base.

Full expressions are similar to many other high level languages. Since there is only one kind of number, there are no rules for mixing types. Instead, there are rules on the scale of expressions. Every expression has a scale. This is derived from the scale of original numbers, the operation performed and in many cases, the value of the variable **scale**. Legal values of the variable **scale** are 0 to the maximum number representable by a C integer.

In the following descriptions of legal expressions, "expr" refers to a complete expression and "var" refers to a simple or an array variable. A simple variable is just a

name

and an array variable is specified as

name[*expr*]

Unless specifically mentioned the scale of the result is the maximum scale of the expressions involved.

- *expr* The result is the negation of the expression.

++ *var* The variable is incremented by one and the new value is the result of the expression.

-- *var* The variable is decremented by one and the new value is the result of the

expression.

var **++** The result of the expression is the value of the variable and then the variable is incremented by one.

var **--** The result of the expression is the value of the variable and then the variable is decremented by one.

expr **+** *expr*
The result of the expression is the sum of the two expressions.

expr **-** *expr*
The result of the expression is the difference of the two expressions.

expr ***** *expr*
The result of the expression is the product of the two expressions.

expr **/** *expr*
The result of the expression is the quotient of the two expressions. The scale of the result is the value of the variable **scale**.

expr **%** *expr*
The result of the expression is the "remainder" and it is computed in the following way. To compute a%b, first a / b is computed to **scale** digits. That result is used to compute a-(a / b)*b to the scale of the maximum of **scale**+*scale*(*b*) and *scale*(*a*). If **scale** is set to zero and both expressions are integers this expression is the integer remainder function.

expr **^** *expr*
The result of the expression is the value of the first raised to the second. The second expression must be an integer. (If the second expression is not an integer, a warning is generated and the expression is truncated to get an integer value.) The scale of the result is **scale** if the exponent is negative. If the exponent is positive the scale of the result is the minimum of the scale of the first expression times the value of the exponent and the maximum of **scale** and the scale of the first expression. (e.g. scale(a^b) = min(scale(a)*b, max(*scale*, scale(a))).) It should be noted that expr^0 will always return the value of 1.

(*expr* **)** This alters the standard precedence to force the evaluation of the expression.

var **=** *expr*
The variable is assigned the value of the expression.

*var op***=** *expr*
This is equivalent to *var* **=** *var op expr* with the exception that the *var* part is evaluated only once. This can make a difference if *var* is an array.

Relational expressions are a special kind of expression that always evaluate to 0 or 1, 0 if the relation is false and 1 if the relation is true. These may appear in any legal expression. (POSIX *bc* requires that relational expressions are used only in **if**, **while**, and **for** statements and that only one relational test may be done in them.) The relational operators are

expr1 **<** *expr2*
The result is 1 if *expr1* is strictly less than *expr2*.

expr1 **<=** *expr2*
The result is 1 if *expr1* is less than or equal to *expr2*.

expr1 **>** *expr2*
The result is 1 if *expr1* is strictly greater than *expr2*.

expr1 **>=** *expr2*
The result is 1 if *expr1* is greater than or equal to *expr2*.

expr1 **==** *expr2*
The result is 1 if *expr1* is equal to *expr2*.

expr1 **!=** *expr2*
The result is 1 if *expr1* is not equal to *expr2*.

Boolean operations are also legal. (POSIX *bc* does *not* have boolean operations). The result of all boolean operations are 0 and 1 (for false and true) as in relational expressions. The boolean operators are:

!*expr* The result is 1 if *expr* is 0.
expr **&&** *expr*
The result is 1 if both expressions are non-zero.
expr **||** *expr*
The result is 1 if either expression is non-zero.

The expression precedence is as follows: (lowest to highest)

Operator	**Association**
||	left associative
&&	left associative
!	nonassociative
Relational (<,>,==,<=,>=)	left associative
=	right associative
+, –	left associative
*, /, %	left associative
^	right associative
unary -	nonassociative
++, --	nonassociative

This precedence was chosen so that POSIX compliant *bc* programs will run correctly. This will cause the use of the relational and logical operators to have some unusual behavior when used with assignment expressions. Consider the expression:

```
a = 3 < 5
```

Most C programmers would assume this would assign the result of "3 < 5" (the value 1) to the variable "a". What this does in *bc* is assign the value 3 to the variable "a" and then compare 3 to 5. It is best to use parenthesis when using relational and logical operators with the assignment operators.

There are a few more special expressions that are provided in *bc*. These have to do with user defined functions and standard functions. They all appear as "*name*(*parameters*)". See the section on functions for user defined functions. The standard functions are:

length(*expression*)
The value of the *length*( ) function is the number of significant digits in the expression.
read( ) The *read*( ) function (an extension) will read a number from the standard input, regardless of where the function occurs. Beware, this can cause problems with the mixing of data and program in the standard input. The best use for this function is in a previously written program that needs input from the user, but never allows program code to be input from the user. The value of the read function is the number read from the standard input using the current value of the variable **ibase** for the conversion base.
scale(*expression*)
The value of the *scale*( ) function is the number of digits after the decimal point in the expression.

sqrt(*expression*)

The value of the *sqrt*( ) function is the square root of the expression. If the expression is negative, a run time error is generated.

STATEMENTS

Statements (as in most algebraic languages) provide the sequencing of expression evaluation. In *bc* statements are executed "as soon as possible." Execution happens when a newline in encountered and there is one or more complete statements. Due to this immediate execution, newlines are very important in *bc*. In fact, both a semicolon and a newline are used as statement separators. An improperly placed newline will cause a syntax error. Because newlines are statement separators, it is possible to hide a newline by using the backslash character. The sequence "\<nl>", where <nl> is the newline appears to *bc* as whitespace instead of a newline. A statement list is a series of statements separated by semicolons and newlines. The following is a list of *bc* statements and what they do: (Things enclosed in brackets ([]) are optional parts of the statement.)

expression

This statement does one of two things. If the expression starts with "*variable assignment ...*", it is considered to be an assignment statement. If the expression is not an assignment statement, the expression is evaluated and printed to the output. After the number is printed, a newline is printed. For example, "a=1" is an assignment statement and "(a=1)" is an expression that has an embedded assignment. All numbers that are printed are printed in the base specified by the variable **obase**. The legal values for **obase** are 2 through {BC_BASE_MAX}. (See the section "LIMITS".) For bases 2 through 16, the usual method of writing numbers is used. For bases greater than 16, *bc* uses a multi-character digit method of printing the numbers where each higher base digit is printed as a base 10 number. The multi-character digits are separated by spaces. Each digit contains the number of characters required to represent the base ten value of "**obase**–1". Since numbers are of arbitrary precision, some numbers may not be printable on a single output line. These long numbers will be split across lines using the "\" as the last character on a line. The maximum number of characters printed per line is 70. Due to the interactive nature of *bc* printing a number cause the side effect of assigning the printed value to the special variable **last**. This allows the user to recover the last value printed without having to retype the expression that printed the number. Assigning to **last** is legal and will overwrite the last printed value with the assigned value. The newly assigned value will remain until the next number is printed or another value is assigned to **last**. (Some installations may allow the use of a single period (.) which is not part of a number as a short hand notation for for **last**.)

string The string is printed to the output. Strings start with a double quote character and contain all characters until the next double quote character. All characters are take literally, including any newline. No newline character is printed after the string.

print *list*

The print statement (an extension) provides another method of output. The *list* is a list of strings and expressions separated by commas. Each string or expression is printed in the order of the list. No terminating newline is printed. Expressions are evaluated and their value is printed and assigned the the variable **last**. Strings in the print statement are printed to the output and may

contain special characters. Special characters start with the backslash character (\). The special characters recognized by *bc* are "a" (alert or bell), "b" (backspace), "f" (form feed), "n" (newline), "r" (carriage return), "q" (double quote), "t" (tab), and "\" (backslash). Any other character following the backslash will be ignored.

{ *statement_list* **}**

This is the compound statement. It allows multiple statements to be grouped together for execution.

if(*expression*) **then** *statement1* [**else** *statement2*]

The **if** statement evaluates the expression and executes *statement1* or *statement2* depending on the value of the expression. If the expression is non-zero, *statement1* is executed. If *statement2* is present and the value of the expression is 0, then *statement2* is executed. (The **else** clause is an extension.)

while(*expression*) *statement*

The **while** statement will execute the *statement* while the expression is non-zero. It evaluates the *expression* before each execution of the *statement*. Termination of the loop is caused by a zero expression value or the execution of a break statement.

for([*expression1*]; [*expression2*]; [*expression3*]) *statement*

The **for** statement controls repeated execution of the *statement*. *Expression1* is evaluated before the loop. *Expression2* is evaluated before each execution of the *statement*. If it is non-zero, the *statement* is evaluated. If it is zero, the loop is terminated. After each execution of the *statement*, *expression3* is evaluated before the reevaluation of *expression2*. If *expression1* or *expression3* are missing, nothing is evaluated at the point they would be evaluated. If *expression2* is missing, it is the same as substituting the value 1 for *expression2*. (The optional expressions are an extension. POSIX *bc* requires all three expressions.) The following is equivalent code for the **for** statement:

```
expression1;
while (expression2) {
   statement;
   expression3;
}
```

break This statement causes a forced exit of the most recent enclosing **while** statement or **for** statement.

continue

The **continue** statement (an extension) causes the most recent enclosing for statement to start the next iteration.

halt The **halt** statement (an extension) is an executed statement that causes the *bc* processor to quit only when it is executed. For example, "if (0 == 1) halt" will not cause *bc* to terminate because the halt is not executed.

return Return the value 0 from a function. (See the section on functions.)

return(*expression*)

Return the value of the expression from a function. (See the section on functions.)

PSEUDO STATEMENTS

These statements are not statements in the traditional sense. They are not executed statements. Their function is performed at "compile" time.

limits Print the local limits enforced by the local version of *bc*. This is an extension.

quit When the quit statement is read, the *bc* processor is terminated, regardless of where the quit statement is found. For example, "if (0 == 1) quit" will cause *bc* to terminate.

warranty
Print a longer warranty notice. This is an extension.

FUNCTIONS

Functions provide a method of defining a computation that can be executed later. Functions in *bc* always compute a value and return it to the caller. Function definitions are "dynamic" in the sense that a function is undefined until a definition is encountered in the input. That definition is then used until another definition function for the same name is encountered. The new definition then replaces the older definition. A function is defined as follows:

define *name* **(***parameters* **) {** *newline*
auto_list statement_list **}**

A function call is just an expression of the form

name(*parameters*)

Parameters are numbers or arrays (an extension). In the function definition, zero or more parameters are defined by listing their names separated by commas. Numbers are only call by value parameters. Arrays are only call by variable. Arrays are specified in the parameter definition by the notation "*name*[]". In the function call, actual parameters are full expressions for number parameters. The same notation is used for passing arrays as for defining array parameters. The named array is passed by variable to the function. Since function definitions are dynamic, parameter numbers and types are checked when a function is called. Any mismatch in number or types of parameters will cause a runtime error. A runtime error will also occur for the call to an undefined function.

The *auto_list* is an optional list of variables that are for "local" use. The syntax of the auto list (if present) is

auto *name*, ... ;

(The semicolon is optional.) Each *name* is the name of an auto variable. Arrays may be specified by using the same notation as used in parameters. These variables have their values pushed onto a stack at the start of the function. The variables are then initialized to zero and used throughout the execution of the function. At function exit, these variables are popped so that the original value (at the time of the function call) of these variables are restored. The parameters are really auto variables that are initialized to a value provided in the function call. Auto variables are different than traditional local variables in the fact that if function A calls function B, B may access function A's auto variables by just using the same name, unless function B has called them auto variables. Due to the fact that auto variables and parameters are pushed onto a stack, *bc* supports recursive functions.

The function body is a list of *bc* statements. Again, statements are separated by semicolons or newlines. To end the function and return a value, use a **return** statement. You don't need an explicit **return**; There is one implied, returning 0, at the end of every function.) There are two versions of the **return** statement:

return returns the value 0 to the calling expression.
return (*expression*
computes the value of the expression and returns that value to the calling expression.

Functions also change the usage of the variable **ibase**. All constants in the function body will be converted using the value of *base* at the time of the function call. Changes of **ibase** will be ignored during the execution of the function except for the standard function *read*( ), which will always use the current value of **ibase** for conversion of numbers.

MATH LIBRARY

If *bc* is invoked with the **–l** option, it loads a math library and sets the default scale to 20. The math functions will calculate their results to the scale set at the time of their call. The math library defines the following functions:

s (*x*) The sine of *x* in radians.
c (*x*) The cosine of *x* in radians.
a (*x*) The arctangent of *x*.
l (*x*) The natural logarithm of *x*.
e (*x*) The exponential function of raising e to the value *x*.
j (*n*,*x*) The Bessel function of integer order *n* of *x*.

EXAMPLES

In *sh*, the following will assign the value of "pi" to the shell variable *pi*:

```
pi=$(echo "scale=10; 4*a(1)" | bc -l)
```

The following is the definition of the exponential function used in the math library. This function is written in POSIX *bc*.

```
scale = 20
/* Uses the fact that e^x = (e^(x/2))^2
   When x is small enough, we use the series:
     e^x = 1 + x + x^2/2! + x^3/3! + ...
*/
define e(x) {
  auto  a, d, e, f, i, m, v, z
  /* Check the sign of x. */
  if (x<0) {
    m = 1
    x = -x
  }
```

```
  /* Precondition x. */
  z = scale;
  scale = 4 + z + .44*x;
  while (x > 1) {
    f += 1;
    x /= 2;
  }
  /* Initialize the variables. */
  v = 1+x
  a = x
  d = 1
  for (i=2; 1; i++) {
    e = (a *= x) / (d *= i)
    if (e == 0) {
      if (f>0) while (f--)  v = v*v;
      scale = z
      if (m) return (1/v);
      return (v/1);
    }
    v += e
  }
}
```

The following code uses the extended features of *bc* to implement a simple program for calculating checkbook balances. This program is best kept in a file so that it can be used many times without having to retype it at every use.

```
scale=2;
print "\nCheck book program!\n"
print "  Remember, deposits are negative transactions.\n"
print "  Exit by a 0 transaction.\n\n"
print "Initial balance? "; bal = read()
bal /= 1
print "\n"

while (1) {
  "current balance = "; bal
  "transaction? "; trans = read()
  if (trans == 0) break;
  bal -= trans
  bal /= 1
}
quit
```

The following is the definition of the recursive factorial function.

```
define f (x) {
  if (x <= 1) return (1);
  return (f(x-1) * x);
}
```

DIFFERENCES

This version of *bc* was implemented from the POSIX P1003.2 / D11 draft and contains several differences and extensions relative to the draft and traditional implementations. It is not implemented in the traditional way using *dc*. This version is a single process which parses and runs a byte code translation of the program. There is an "undocumented" option (**–c**) that causes the program to output the byte code to the standard output instead of running it. It was mainly used for debugging the parser and preparing the math library.

A major source of differences is extensions, where a feature is extended to add more functionality and additions, where new features are added.

To disable extensions and make this *bc* behave as specified by POSIX, use the **–s** option. The following is the list of differences and extensions.

LANG environment
: This version does not conform to the POSIX standard in the processing of the *LANG* environment variable and all environment variables starting with *LC_*.

Names
: Traditional and POSIX *bc* have single letter names for functions, variables and arrays. They have been extended to be multi-character names that start with a letter and may contain letters, numbers and the underscore character.

Strings
: Strings are not allowed to contain NUL characters. POSIX says all characters must be included in strings.

last
: POSIX *bc* does not have a **last** variable. Some implementations of *bc* use the dot (.) in a similar way.

Comparisons
: POSIX *bc* allows comparisons only in the **if** statement, the **while** statement, and the second expression of the **for** statement. Also, only one relational operation is allowed in each of those statements.

if statement, **else** clause
: POSIX *bc* does not have an **else** clause.

for statement
: POSIX *bc* requires all expressions to be present in the **for** statement.

&&, | |, !
: POSIX *bc* does not have the logical operators.

read() function
: POSIX *bc* does not have a *read*( ) function.

print statement
: POSIX *bc* does not have a **print** statement.

continue statement
: POSIX *bc* does not have a **continue** statement.

Array parameters
: POSIX *bc* does not have array parameters. Other implementations of *bc* may have call by value array parameters.

=+, =-, =*, =/, =%, =^
: POSIX *bc* does not require these "old style" assignment operators to be defined. This version may allow these "old style" assignments. Use the **limits** statement to see if the installed version supports them. If it does support the "old style" assignment operators, the statement "a =- 1" will decrement a by 1 instead of setting a to the value –1.

Spaces in numbers
: Other implementations of *bc* allow spaces in numbers. For example, "x=1 3"

would assign the value 13 to the variable x. The same statement would cause a syntax error in this version of *bc*.

Errors and execution

This implementation varies from other implementations in terms of what code will be executed when syntax and other errors are found in the program. If a syntax error is found in a function definition, error recovery tries to find the beginning of a statement and continue to parse the function. Once a syntax error is found in the function, the function will not be callable and becomes undefined. Syntax errors in the interactive execution code will invalidate the current execution block. The execution block is terminated by an end of line that appears after a complete sequence of statements. For example,

```
a = 1
b = 2
```

has two execution blocks and

```
{ a = 1
  b = 2 }
```

has one execution block. Any runtime error will terminate the execution of the current execution block. A runtime warning will not terminate the current execution block.

Interrupts

During an interactive session, the SIGINT signal (usually generated by the control-C character from the terminal) will cause execution of the current execution block to be interrupted. It will display a "runtime" error indicating which function was interrupted. After all runtime structures have been cleaned up, a message will be printed to notify the user that *bc* is ready for more input. All previously defined functions remain defined and the value of all non-auto variables are the value at the point of interruption. All auto variables and function parameters are removed during the clean up process. During a non-interactive session, the SIGINT signal will terminate the entire run of *bc*.

LIMITS

The following are the limits currently in place for this *bc* processor. Some of them may have been changed by an installation. Use the **limits** statement to see the actual values.

{BC_BASE_MAX}

The maximum output base is currently set at 999. The maximum input base is 16.

{BC_DIM_MAX}

This is currently an arbitrary limit of 65535 as distributed. Your installation may be different.

{BC_SCALE_MAX}

The number of digits after the decimal point is limited to {INT_MAX} digits. Also, the number of digits before the decimal point is limited to {INT_MAX} digits.

{BC_STRING_MAX}

The limit on the number of characters in a string is {INT_MAX} characters.

Exponent

The value of the exponent in the raise operation (^) is limited to {LONG_MAX}.

Multiply
: The multiply routine may yield incorrect results if a number has more than LONG_MAX / 90 total digits. For 32 bit longs, this number is 23,860,929 digits.

Code
: Each function and the "main" program are limited to 16384 bytes of compiled byte code each. This limit ({BC_MAX_SEGS}) can be easily changed to have more than 16 segments of 1024 bytes.

Variable
: The current limit on the number of unique names is 32767 for each of simple variables, arrays and functions.

FILES

In most installations, *bc* is completely self-contained. Where executable size is of importance or the C compiler does not deal with very long strings, *bc* will read the standard math library from the file **$OPENNT_ROOT/share/libmath.b**. (On other operating systems, the package may be installed in **/usr/local/lib/libmath.b** or **/lib/libmath.b**.)

DIAGNOSTICS

If any file on the command line can not be opened, *bc* will report that the file is unavailable and terminate. Also, there are compile and run time diagnostics that should be self-explanatory.

VERSION

This man page documents GNU *bc* version 1.03.

BUGS

Error recovery is not very good yet.

AUTHOR

Philip A. Nelson phil@cs.wwu.edu

ACKNOWLEDGEMENTS

The author would like to thank Steve Sommars (Steve.Sommars@att.com) for his extensive help in testing the implementation. Many great suggestions were given. This is a much better product due to his involvement.

NAME

bdftopcf – convert font from Bitmap Distribution Format to Portable Compiled Format

SYNOPSIS

`bdftopcf [-option ...] font-file.bdf`

DESCRIPTION

Bdftopcf is the release 5 font compiler. Fonts in Portable Compiled Format can be read by any architecture, although the file is structured to allow one particular architecture to read them directly without reformatting. This allows fast reading on the appropriate machine, but the files are still portable (but read more slowly) on other machines.

OPTIONS

–p*n* Sets the font glyph padding. Each glyph in the font will have each scanline padded in to a multiple of *n* bytes, where *n* is 1, 2, 4 or 8.

–u*n* Sets the font scanline unit. When the font bit order is different from the font byte order, the scanline unit *n* describes what unit of data (in bytes) are to be swapped; the unit *i* can be 1, 2 or 4 bytes.

–m Sets the font bit order to MSB (most significant bit) first. Bits for each glyph will be placed in this order; i.e. the left most bit on the screen will be in the highest valued bit in each unit.

–l Sets the font bit order to LSB (least significant bit) first. The left most bit on the screen will be in the lowest valued bit in each unit.

–M Sets the font byte order to MSB first. All multi-byte data in the file (metrics, bitmaps and everything else) will be written most significant byte first.

–L Sets the font byte order to LSB first. All multi-byte data in the file (metrics, bitmaps and everything else) will be written least significant byte first.

–t When this option is specified, *bdftopcf* will convert fonts into "terminal" fonts when possible. A terminal font has each glyph image padded to the same size; the X server can usually render these types of fonts more quickly.

–i This option inhibits the normal computation of ink metrics. When a font has glyph images which do not fill the bitmap image (i.e. the "on" pixels don't extend to the edges of the metrics) *bdftopcf* computes the actual ink metrics and places them in the .pcf file; the –t option inhibits this behaviour.

–o*output-file-name*
By default *bdftopcf* writes the pcf file to standard output; this option gives the name of a file to be used instead.

SEE ALSO

X

COPYRIGHT

Copyright 1991, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Keith Packard, MIT X Consortium

NAME

bitmap, bmtoa, atobm – bitmap editor and converter utilities for the X Window System

SYNOPSIS

`bitmap [–options ...] [filename ] [basename]`

`bmtoa [–chars ...] [filename]`

`atobm [–chars cc] [–name variable]`
`[–xhot number] [–yhot number] [filename]`

DESCRIPTION

The *bitmap* program is a rudimentary tool for creating or editing rectangular images made up of 1's and 0's. Bitmaps are used in X for defining clipping regions, cursor shapes, icon shapes, and tile and stipple patterns.

The *bmtoa* and *atobm* filters convert *bitmap* files (FILE FORMAT) to and from ASCII strings. They are most commonly used to quickly print out bitmaps and to generate versions for including in text.

COMMAND LINE OPTIONS

Bitmap supports the standard X Toolkit command line arguments (see *X*). The following additional arguments are supported as well.

–size *WIDTHxHEIGHT*
Specifies size of the grid in squares.

–sw *dimension*
Specifies the width of squares in pixels.

–sh *dimension*
Specifies the height of squares in pixels.

–gt *dimension*
Grid tolerance. If the square dimensions fall below the specified value, grid will be automatically turned off.

–grid on|off
+grid Turns on or off the grid lines.

–axes on|off
+axes Turns on or off the major axes.

–dashed on|off
+dashed
Turns on or off dashing for the frame and grid lines.

–stippled on|off
+stippled
Turns on or off stippling of highlighted squares.

–proportional on|off
+proportional
Turns proportional mode on or off. If proportional mode is on, square width is equal to square height. If proportional mode is off, *bitmap* will use the smaller square dimension, if they were initially different.

–dashes *filename*
Specifies the bitmap to be used as a stipple for dashing.

–stipple *filename*
Specifies the bitmap to be used as a stipple for highlighting.

–hl *color*
Specifies the color used for highlighting.

–fr *color*
Specifies the color used for the frame and grid lines.

filename
Specifies the bitmap to be initially loaded into the program. If the file does not exist, *bitmap* will assume it is a new file.

basename
Specifies the basename to be used in the C code output file. If it is different than the basename in the working file, *bitmap* will change it when saving the file.

Bmtoa accepts the following option:

–chars *cc*
This option specifies the pair of characters to use in the string version of the bitmap. The first character is used for 0 bits and the second character is used for 1 bits. The default is to use dashes (–) for 0's and sharp signs (#) for 1's.

Atobm accepts the following options:

–chars *cc*
This option specifies the pair of characters to use when converting string bitmaps into arrays of numbers. The first character represents a 0 bit and the second character represents a 1 bit. The default is to use dashes (–) for 0's and sharp signs (#) for 1's.

–name *variable*
This option specifies the variable name to be used when writing out the bitmap file. The default is to use the basename of the *filename* command line argument or leave it blank if the standard input is read.

–xhot *number*
This option specifies the X coordinate of the hotspot. Only positive values are allowed. By default, no hotspot information is included.

–yhot *number*
This option specifies the Y coordinate of the hotspot. Only positive values are allowed. By default, no hotspot information is included.

USAGE

Bitmap displays grid in which each square represents a single bit in the picture being edited. Actual size of the bitmap image, as it would appear normally and inverted, can be obtained by pressing the **Meta-I** key. You are free to move the image popup out of the way to continue editing. Pressing the left mouse button in the popup window or **Meta-I** again will remove the real size bitmap image.

If the bitmap is to be used for defining a cursor, one of the squares in the images may be designated as the hot spot. This determines where the cursor is actually pointing. For cursors with sharp tips (such as arrows or fingers), this is usually at the end of the tip;

for symmetric cursors (such as crosses or bullseyes), this is usually at the center.

Bitmaps are stored as small C code fragments suitable for including in applications. They provide an array of bits as well as symbolic constants giving the width, height, and hot spot (if specified) that may be used in creating cursors, icons, and tiles.

EDITING

To edit a bitmap image simply click on one of the buttons with drawing commands (**Point**, **Curve**, **Line**, **Rectangle**, etc.) and move the pointer into the bitmap grid window. Press one of the buttons on your mouse and the appropriate action will take place. You can either set, clear or invert the gird squares. Setting a grid square corresponds to setting a bit in the bitmap image to 1. Clearing a grid square corresponds to setting a bit in the bitmap image to 0. Inverting a grid square corresponds to changing a bit in the bitmap image from 0 to 1 or 1 to 0, depending what its previous state was. The default behavior of mouse buttons is as specified below.

MouseButton1	Set
MouseButton2	Invert
MouseButton3	Clear
MouseButton4	Clear
MouseButton5	Clear

This default behavior can be changed by setting the button function resources. An example is provided below.

```
bitmap*button1Function: Set
bitmap*button2Function: Clear
bitmap*button3Function: Invert
```
etc.

The button function applies to all drawing commands, including copying, moving and pasting, flood filling and setting the hot spot.

DRAWING COMMANDS

Here is the list of drawing commands accessible through the buttons at the left side of the application's window. Some commands can be aborted by pressing A inside the bitmap window, allowing the user to select different guiding points where applicable.

Clear This command clears all bits in the bitmap image. The grid squares will be set to the background color. Pressing C inside the bitmap window has the same effect.

Set This command sets all bits in the bitmap image. The grid squares will be set to the foreground color. Pressing S inside the bitmap window has the same effect.

Invert This command inverts all bits in the bitmap image. The grid squares will be inverted appropriately. Pressing I inside the bitmap window has the same effect.

Mark This command is used to mark an area of the grid by dragging out a rectangular shape in the highlighting color. Once the area is marked, it can be operated on by a number of commands (see **Up, Down, Left, Right, Rotate, Flip, Cut,** etc.) Only one marked area can be present at any time. If you attempt to mark another area, the old mark will vanish. The same effect can be achieved by pressing **Shift-MouseButton1** and dragging out a rectangle in the grid window. Pressing **Shift-MouseButton2** will mark the entire grid area.

Unmark

This command will cause the marked area to vanish. The same effect can be achieved by pressing **Shift-MouseButton3**.

Copy This command is used to copy an area of the grid from one location to another. If there is no marked grid area displayed, **Copy** behaves just like **Mark** described above. Once there is a marked grid area displayed in the highlighting color, this command has two alternative behaviors. If you click a mouse button inside the marked area, you will be able to drag the rectangle that represents the marked area to the desired location. After you release the mouse button, the area will be copied. If you click outside the marked area, **Copy** will assume that you wish to mark a different region of the bitmap image, thus it will behave like **Mark** again.

Move This command is used to move an area of the grid from one location to another. Its behavior resembles the behavior of **Copy** command, except that the marked area will be moved instead of copied.

Flip Horizontally

This command will flip the bitmap image with respect to the horizontal axes. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing F inside the bitmap window has the same effect.

Up This command moves the bitmap image one pixel up. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing UpArrow inside the bitmap window has the same effect.

Flip Vertically

This command will flip the bitmap image with respect to the vertical axes. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing V inside the bitmap window has the same effect.

Left This command moves the bitmap image one pixel to the left. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing LeftArrow inside the bitmap window has the same effect.

Fold This command will fold the bitmap image so that the opposite corners become adjacent. This is useful when creating bitmap images for tiling. Pressing F inside the bitmap window has the same effect.

Right This command moves the bitmap image one pixel to the right. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing RightArrow inside the bitmap window has the same effect.

Rotate Left

This command rotates the bitmap image 90 degrees to the left (counter clockwise.) If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing L inside the bitmap window has the same effect.

Down This command moves the bitmap image one pixel down. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing DownArrow inside the bitmap window has the same effect.

Rotate Right

This command rotates the bitmap image 90 degrees to the right (clockwise.) If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing R inside the bitmap window has the same effect.

Point This command will change the grid squares underneath the mouse pointer if a mouse button is being pressed down. If you drag the mouse button continuously, the line may not be continuous, depending on the speed of your system and frequency of mouse motion events.

Curve This command will change the grid squares underneath the mouse pointer if a

mouse button is being pressed down. If you drag the mouse button continuously, it will make sure that the line is continuous. If your system is slow or bitmap*receives*very behave quite strangely.

Line This command will change the gird squares in a line between two squares. Once you press a mouse button in the grid window, *bitmap* will highlight the line from the square where the mouse button was initially pressed to the square where the mouse pointer is located. By releasing the mouse button you will cause the change to take effect, and the highlighted line will disappear.

Rectangle

This command will change the gird squares in a rectangle between two squares. Once you press a mouse button in the grid window, *bitmap*will highlight the rectangle from the square where the mouse button was initially pressed to the square where the mouse pointer is located. By releasing the mouse button you will cause the change to take effect, and the highlighted rectangle will disappear.

Filled Rectangle

This command is identical to **Rectangle**, except at the end the rectangle will be filled rather than outlined.

Circle This command will change the gird squares in a circle between two squares. Once you press a mouse button in the grid window, *bitmap* will highlight the circle from the square where the mouse button was initially pressed to the square where the mouse pointer is located. By releasing the mouse button you will cause the change to take effect, and the highlighted circle will disappear.

Filled Circle

This command is identical to **Circle**, except at the end the circle will be filled rather than outlined.

Flood Fill

This command will flood fill the connected area underneath the mouse pointer when you click on the desired square. Diagonally adjacent squares are not considered to be connected.

Set Hot Spot

This command designates one square in the grid as the hot spot if this bitmap image is to be used for defining a cursor. Pressing a mouse button in the desired square will cause a diamond shape to be displayed.

Clear Hot Spot

This command removes any designated hot spot from the bitmap image.

Undo This command will undo the last executed command. It has depth one, that is, pressing **Undo** after **Undo** will undo itself.

FILE MENU

The File menu commands can be accessed by pressing the File button and selecting the appropriate menu entry, or by pressing Ctrl key with another key. These commands deal with files and global bitmap parameters, such as size, basename, filename etc.

New This command will clear the editing area and prompt for the name of the new file to be edited. It will not load in the new file.

Load This command is used to load a new bitmap file into the bitmap editor. If the current image has not been saved, user will be asked whether to save or ignore the changes. The editor can edit only one file at a time. If you need interactive editing, run a number of editors and use cut and paste mechanism as described below.

Insert

This command is used to insert a bitmap file into the image being currently edited. After being prompted for the filename, click inside the grid window and drag the outlined rectangle to the location where you want to insert the new file.

Save This command will save the bitmap image. It will not prompt for the filename unless it is said to be <none>. If you leave the filename undesignated or –, the output will be piped to stdout.

Save As
This command will save the bitmap image after prompting for a new filename. It should be used if you want to change the filename.

Resize This command is used to resize the editing area to the new number of pixels. The size should be entered in the WIDTHxHEIGHT format. The information in the image being edited will not be lost unless the new size is smaller that the current image size. The editor was not designed to edit huge files.

Rescale This command is used to rescale the editing area to the new width and height. The size should be entered in the WIDTHxHEIGHT format. It will not do antialiasing and information will be lost if you rescale to the smaller sizes. Feel free to add you own algorithms for better rescaling.

Filename
This command is used to change the filename without changing the basename nor saving the file. If you specify – for a filename, the output will be piped to stdout.

Basename
This command is used to change the basename, if a different one from the specified filename is desired.

Quit This command will terminate the bitmap application. If the file was not saved, user will be prompted and asked whether to save the image or not. This command is preferred over killing the process.

EDIT MENU

The Edit menu commands can be accessed by pressing the Edit button and selecting the appropriate menu entry, or by pressing Meta key with another key. These commands deal with editing facilities such as grid, axes, zooming, cut and paste, etc.

Image This command will display the image being edited and its inverse in its actual size in a separate window. The window can be moved away to continue with editing. Pressing the left mouse button in the image window will cause it to disappear from the screen.

Grid This command controls the grid in the editing area. If the grid spacing is below the value specified by gridTolerance resource (8 by default), the grid will be automatically turned off. It can be enforced by explicitly activating this command.

Dashed This command controls the stipple for drawing the grid lines. The stipple specified by dashes resource can be turned on or off by activating this command.

Axes This command controls the highlighting of the main axes of the image being edited. The actual lines are not part of the image. They are provided to aid user when constructing symmetrical images, or whenever having the main axes highlighted helps your editing.

Stippled
This command controls the stippling of the highlighted areas of the bitmap image. The stipple specified by stipple resource can be turned on or off by

activating this command.

Proportional
This command controls the proportional mode. If the proportional mode is on, width and height of all image squares are forced to be equal, regardless of the proportions of the bitmap window.

Zoom This command controls the zoom mode. If there is a marked area of the image already displayed, bitmap will automatically zoom into it. Otherwise, user will have to highlight an area to be edited in the zoom mode and bitmap will automatically switch into it. One can use all the editing commands and other utilities in the zoom mode. When you zoom out, undo command will undo the whole zoom session.

Cut This commands cuts the contents of the highlighted image area into the internal cut and paste buffer.

Copy This command copies the contents of the highlighted image area into the internal cut and paste buffer.

Paste This command will check if there are any other bitmap applications with a highlighted image area, or if there is something in the internal cut and paste buffer and copy it to the image. To place the copied image, click in the editing window and drag the outlined image to the position where you want to place i, and then release the button.

CUT AND PASTE

Bitmap supports two cut and paste mechanisms; the internal cut and paste and the global X selection cut and paste. The internal cut and paste is used when executing copy and move drawing commands and also cut and copy commands from the edit menu. The global X selection cut and paste is used whenever there is a highlighted area of a bitmap image displayed anywhere on the screen. To copy a part of image from another bitmap editor simply highlight the desired area by using the Mark command or pressing the shift key and dragging the area with the left mouse button. When the selected area becomes highlighted, any other applications (such as xterm, etc.) that use primary selection will discard their selection values and unhighlight the appropriate information. Now, use the Paste command for the Edit menu or control mouse button to copy the selected part of image into another (or the same) bitmap application. If you attempt to do this without a visible highlighted image area, the bitmap will fall back to the internal cut and paste buffer and paste whatever was there stored at the moment.

WIDGETS

Below is the widget structure of the *bitmap* application. Indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name. All widgets except the bitmap widget are from the standard Athena widget set.

```
Bitmap bitmap
       TransientShell image
              Box box
                     Label normalImage
                     Label invertedImage
       TransientShell input
              Dialog dialog
                     Command okay
                     Command cancel
       TransientShell error
              Dialog dialog
                     Command abort
                     Command retry
       TransientShell qsave
              Dialog dialog
                     Command yes
                     Command no
                     Command cancel
       Paned parent
              Form formy
                     MenuButton fileButton
                     SimpleMenu fileMenu
                            SmeBSB  new
                            SmeBSB  load
                            SmeBSB  insert
                            SmeBSB  save
                            SmeBSB  saveAs
                            SmeBSB  resize
                            SmeBSB  rescale
                            SmeBSB  filename
                            SmeBSB  basename
                            SmeLine line
                            SmeBSB  quit
                     MenuButton editButton
                     SimpleMenu editMenu
                            SmeBSB  image
                            SmeBSB  grid
                            SmeBSB  dashed
                            SmeBSB  axes
                            SmeBSB  stippled
                            SmeBSB  proportional
                            SmeBSB  zoom
                            SmeLine line
                            SmeBSB  cut
                            SmeBSB  copy
                            SmeBSB  paste
                     Label status
              Pane pane
                     Bitmap bitmap
                     Form form
                            Command clear
```

```
Command set
Command invert
Toggle  mark
Command unmark
Toggle  copy
Toggle  move
Command flipHoriz
Command up
Command flipVert
Command left
Command fold
Command right
Command rotateLeft
Command down
Command rotateRight
Toggle  point
Toggle  curve
Toggle  line
Toggle  rectangle
Toggle  filledRectangle
Toggle  circle
Toggle  filledCircle
Toggle  floodFill
Toggle  setHotSpot
Command clearHotSpot
Command undo
```

COLORS

If you would like *bitmap* to be viewable in color, include the following in the #ifdef COLOR section of the file you read with *xrdb*:

```
*customization:    -color
```

This will cause *bitmap* to pick up the colors in the app-defaults color customization file: **$OPENNT_ROOT/usr/lib/X11/app-defaults/Bitmap-color**.

BITMAP WIDGET

Bitmap widget is a stand-alone widget for editing raster images. It is not designed to edit large images, although it may be used in that purpose as well. It can be freely incorporated with other applications and used as a standard editing tool. The following are the resources provided by the bitmap widget.

```
Bitmap Widget

Header file         Bitmap.h
Class               bitmapWidgetClass
Class Name          Bitmap
Superclass          Bitmap
```

All the Simple Widget resources plus . . .

Name	Class	Type	Default Value
foreground	Foreground	Pixel	XtDefaultForeground
highlight	Highlight	Pixel	XtDefaultForeground
framing	Framing	Pixel	XtDefaultForeground
gridTolerance	GridTolerance	Dimension	8
size	Size	String	32x32
dashed	Dashed	Boolean	True
grid	Grid	Boolean	True
stippled	Stippled	Boolean	True
proportional	Proportional	Boolean	True
axes	Axes	Boolean	False
squareWidth	SquareWidth	Dimension	16
squareHeight	SquareHeight	Dimension	16
margin	Margin	Dimension	16
xHot	XHot	Position	NotSet (–1)
yHot	YHot	Position	NotSet (–1)
button1Function	Button1Function	DrawingFunction	Set
button2Function	Button2Function	DrawingFunction	Invert
button3Function	Button3Function	DrawingFunction	Clear
button4Function	Button4Function	DrawingFunction	Invert
button5Function	Button5Function	DrawingFunction	Invert
filename	Filename	String	None ("")
basename	Basename	String	None ("")

Author: Davor Matic, X Consortium

NAME

bp – boiler-plate generator

SYNOPSIS

```
bp [-d dirname] boiler-plate-name
bp [-x | [-d dirname] -X]
```

DESCRIPTION

The *bp* utility writes a boiler-plate to standard-output as specified by the *boiler-plate-name* operand.

The options are as follows:

–d *dirname*
: Search for boiler-plates stored in directory *dirname*, rather than in **$HOME/bpdir**.

–X
: List boiler-plates found in the boiler plate directory.

–x
: Output an explanation message.

By default, the *bp* utility will search for boiler-plates in **$HOME/bpdir**. If the environment variable *BPDIR* is set to a directory name, this overrides the default. The **–d** option takes precedence if it is on the command line. The value for *BPDIR* must end with a trailing slash character (but an argument to **–d** need not).

BOILER-PLATES

Boiler-plates are great for improving consistency across similar types of files, for improving productivity when working with known templates, and for reducing retyping errors. There are a set of simple macros that can be included in the boiler-plates to "personalize" the output; see "Macros" below.

Boiler-plates all start with a line that describes the boiler-plate, and is not output as part of the writing to standard-output. This line begins with the four characters `#BP\t`, where `\t` represents a tab. (This can be used in a **magic** file to identify boiler-plates.) Everything else on the line is a simple one-line description of the boiler-plate contents that is written to standard-output when the **–X** flag is used.

Boiler-plates can be used for anything, including simple look-up files for information (e.g. postal codes, the table of elements, macro names, office phone extensions). A sample of the boiler-plates used in the Softway Systems development environment include:

DEBUG	DEBUG macro definition
SSI	List of SSI_ feature test macros.
api.c	Simple function in C (with a test-program framework).
cleanup	Clean-up Checklist for development work in a directory
codeids	Copyright and RCSid character strings.
header.h	Basic header file with guards and copyright notice.
makefile	Simple makefile template with standard targets.
makefile.lib	Makefile for library with simple targets.

makefile.pub	Makefile for a publications directory.
modcopyr.c	Standard Softway copyright for ported source.
modcopyr.mm	Standard Softway copyright for translated man page.
newcopyr.c	Standard Softway copyright for new C source.
newcopyr.sh	Standard Softway copyright for new shell or make source.
setvbuf	Unbuffer calls for stdout and stderr.
sutil.c	Simple utility template for filter program.
testdata	Text file to be used as test data.
util.c	Simple utility template.
util.man	Template man page file (troff man macros).
util.sh	Shell script template w. options, help, and copyright

If boiler-plates are broken out into types across different directories, then *bp* with a specific directory name can be aliased to a more appropriate command name.

```
$ alias lookup="bp -d //X/info"
$ lookup -X
//X/info
Elements            Table of Eeements
ZipCodes            Area zip codes
Suppliers           List of contact numbers.
```

Macros

A macro is a string which *bp* replaces with some value from the user's environment. Macros begin with the @ character; most macros are only two characters long. The macros deal with time, the user's name, environment variables, and shell script output:

Meaning	**Macro**	**Example**
An @ symbol	**@@**	**@**
Weekday name	**@A**	**Tuesday**
Weekday name (abbr)	**@a**	**Tue**
Month	**@B**	**December**
Month (abbr)	**@b**	**Dec**
Date & time	**@C**	**Tue Dec 30 15:24:14 1997**
Date & time	**@c**	**12/30/97 15:24:14**
Date (numeric)	**@D**	**12/30/97**
Day (decimal)	**@d**	**30**
Month (decimal)	**@m**	**12**
Hour (24-hour)	**@H**	**15**
Hour (12-hour)	**@I**	**03**
AM / PM	**@p**	**PM**
Time (24-hour)	**@R**	**15:24**

Full year	**@Y**	**1997**
Abbrev. year	**@y**	**97**
Username, no domain	**@U**	**dana**
Username with domain	**@u**	**ACCTNG+dana**
Environment variable	**@(***VAR***)**	
Command output	**@{***command***}**	

A meaningless macro (for example, one that names a non-existent environment variable or uses an unused letter such as Q) is output without change.

DIAGNOSTICS

The *bp* utility exits 0 on success, and >0 if an error occurs.

EXAMPLE

The following simple boilerplate presents a simple shell script:

```
#BP    Shell script template
#
# Utility name: name
#
# Created by: @U on @C
# Copyright @Y Your Company Name
#
CMD=$(basename $0)
USAGE="usage: $CMD [-f arg] ..."

while getopts f: opt
do
       case "$opt" in
       'f') echo "Option: " $OPTARG
            ;;
       '?') echo $USAGE
            exit 1;;
       esac
done

shift $OPTIND-1

for arg in $@
do
       echo $arg
done
```

NAME

c89, cc – compiler interface

SYNOPSIS

```
c89 [-cEgMOPSsu] [-D name[=value]] ... [-I directory] ...
    [-L directory] ... [-N nostdlib|nostdinc|nostdc]
    [-o outfile] [-U name] [-X CL_options]
    [-Y LINK_options]  ... operand ...

cc [-cEgMOPSsu] [-D name[=value]] ... [-I directory] ...
   [-L directory] ... [-M] [-N nostdlib] [-N nostdinc]
   [-o outfile] [-U name] [-X CL_options]
   [-Y LINK_options]  ... operand ...
```

DESCRIPTION

The *c89* utility is an interface to the system C compiler (in this case, the Microsoft Visual C / C++ compiler). The *operand*s are either the names of files to be compiled or linked (in which case they will have the file extensions **.c** or **.obj**) or they indicate libraries containing modules to be linked (in which case they will begin with the prefix **–l**).

c89 can also be invoked as *cc*. If invoked as *cc*, it behaves differently in these ways:

- **–L** and **–o** become operands, not options: they can be in any position in the command line.
- The option **–D unix** is passed, so the macro `unix` is defined.
- Compiler extensions to the ANSI C standard (such as additional keywords) are allowed (*c89* does not allow extensions.)

c89 and *cc* accept the following options:

–c Compile but do not link, and don't remove any object files.

–D *name***[=***value***]**
Define *name* as if the **#define** keyword had been used. If a *value* is given, it is assigned to *name*; otherwise, *name* has a value of 1. You can specify more than one **–D** option.

–E Expand all C preprocessor directives and copy the C source files to standard output.

–g Produce symbolic information in the object or executable files.

–I *directory*
Search *directory* for header files before looking in the usual places. You can specify more than one **–I** option.

–L *directory*
Search *directory* for library files before looking in the usual places. You can specify more than one **–L** option.

–M Produce a linker map file. This option has no effect if you also specified **–c**, **–E**, or **–P**.

–N *type* Disable automatic inclusion of certain files, depending upon the value of *type*:

nostdc Disable ANSI-only mode in the Visual C++ compiler and allows use of Microsoft extensions. By default, the compiler is run in ANSI C mode.

nostdinc
Disable automatic inclusion of standard header files.

nostdlib
Disable automatic inclusion of standard library files.

–o *outfile*
Use the pathname *outfile* instead of the default name **a.out**. This option cannot be used with **–c**.

–O Optimize the code that is generated for the output file.

–P Preprocess the C source file and write the results to a file with a **.i** extension in place of **.c**.

–S Create an assembler listing; the format and file extension depend upon the underlying compiler.

–s Strip symbolic names and other extraneous information from the object file or executable.

–U *name*
"Undefine" *name* as if the **#undefine** keyword had been used. This takes precedence over any **–D** options given on the command line.

–u Undefine all of the pre-defined macros.

–X *CL_options*
Passes the specified *CL_options* directly to the **CL.EXE** program.

–Y *LINK_options*
Passes the specified *LINK_options* directly to the **LINK.EXE** program.

Libraries can also be specified by using the correct *operand*; for example, **–l m** specifies the math library. (For compatibility with older systems, the space between the **–l** and the library specifier is not required.) The order of operands is important; libraries are searched in the order they appear on the command line.

C89 understands the following **–l** library operands:

–l c Include the standard C library. When the linker is invoked, this library is opened last.

–l l Include the *lex* library.

–l m Include the math library.

–l y Include the *yacc* library.

C89 "knows" the default locations of the header files and the libraries, normally **$OPENNT_ROOT/usr/include** and **$OPENNT_ROOT/usr/lib**. It also "knows" about the default libraries, and includes them in the correct order.

This implementation **#define**s the following macros to be 1: `_POSIX_` and `__OPENNT`. When invoked as *cc*, it also **#define**s `unix`.

FILES

C89 makes use of the following files:

libc.a

INTERIX implementations of C run-time functions. This is placed before **libcpsx.a** so the INTERIX implementations of functions are picked up.

libcpsx.a

Microsoft Visual C / C++ C run-time library for POSIX.

psxdll.a

INTERIX POSIX subsystem entry points. This normally comes last in the command line.

psxrtl.a

C run-time startup code. This is placed before **libcpsx.a** in the final command line.

NOTES

This implementation of *c89* is an interface to the Microsoft Visual C++ 4.0 compiler (**cl.exe**) and linker (**link.exe**). Because those programs don't understand the POSIX filename format, *c89* explicitly unsets the *LIB* and *INCLUDE* environment variables before running. If you need to search other directories for libraries and header files, use the **–L** and **–I** options, respectively.

The following file extensions are valid in operands:

.a Library file.

.c C source file.

.lib Library file.

.o Object file.

.obj Object file.

ENVIRONMENT VARIABLES

The *c89* utility makes use of the following environment variables, if set:

C89_COMPILER

The compiler's complete pathname. Case is significant (for example, `//C/MSVC40/BIN/CL.EXE`). Useful if you don't want to include **//C/MSVC40/BIN** in your *PATH*.

C89_ECHO

If this variable is set to 1, *c89* displays the command lines it executes to run the compiler and linker.

C89_LINKER

The linker's complete pathname. Case is significant (for example, `//C/MSVC40/BIN/LINK.EXE`). Useful if you don't want to include **//C/MSVC40/BIN** in your *PATH*.

DIAGNOSTICS

C89 can return one of the following exit values:

0 Success.

>0 An error occurred.

NOTES

These interfaces work with the Microsoft Visual C / C++ compiler, not with *gcc*.

SEE ALSO

gcc

NAME

cal – displays a calendar

SYNOPSIS

`cal [ -jy] [`*`month`* `[ `*`year`*`]]`

DESCRIPTION

Cal displays a simple calendar. If arguments are not specified, the current month is displayed. The options are as follows:

–j Display Julian dates (days one-based, numbered from January 1).

–y Display a calendar for the current year.

A single parameter specifies the year (1 – 9999) to be displayed; note the year must be fully specified: **cal 89** will *not* display a calendar for 1989. Two parameters denote the month (1 - 12) and year. If no parameters are specified, the current month's calendar is displayed.

A year starts on Jan 1.

The Gregorian Reformation is assumed to have occurred in 1752 on the 3rd of September. By this time, most countries had recognized the reformation (although a few did not recognize it until the early 1900's.) Ten days following that date were eliminated by the reformation, so the calendar for that month is a bit unusual.

SEE ALSO

calendar

NAME

calendar – reminder service

SYNOPSIS

`calendar`

DESCRIPTION

Calendar checks the current directory for a file named **calendar** and displays lines that begin with either today's date or tomorrow's. On Fridays, events on Friday through Monday are displayed.

Lines should begin with a month and day. They may be entered in almost any format, either numeric or as character strings. A single asterisk (''*'') matches every month. A day without a month matches that day of every week. A month without a day matches the first of that month. Two numbers default to the month followed by the day. Lines with leading tabs default to the last entered date, allowing multiple line specifications for a single date. By convention, dates followed by an asterisk are not fixed, i.e., change from year to year.

The ''calendar'' file is preprocessed by *cpp*, allowing the inclusion of shared files such as company holidays or meetings. If the shared file is not referenced by a full pathname, *cpp* searches in the current (or home) directory first, and then in the directory **$OPENNT_ROOT/usr/share/calendar**. Empty lines and lines protected by the C commenting syntax (/* ... */) are ignored.

Some possible calendar entries:

```
#include    <calendar.usholiday>
#include    <calendar.birthday>
6/15        ... June 15 (if ambiguous, will default to month/day).
Jun. 15     ... June 15.
15 June     ... June 15.
Thursday    ... Every Thursday.
June        ... Every June 1st.
15 *        ... 15th of every month.
```

FILES

The following default calendar files are provided:

calendar.birthday
: Births and deaths of famous (and not-so-famous) people.

calendar.christian
: Christian holidays. This calendar should be updated yearly by the local system administrator so that roving holidays are set correctly for the current year.

calendar.computer
: Days of special significance to computer people.

calendar.history
: Everything else, mostly U. S. historical events.

calendar.holiday
: Other holidays, including the not-well-known, obscure, and *really* obscure.

calendar.judaic
: Jewish holidays. This calendar should be updated yearly by the local system administrator so that roving holidays are set correctly for the current year.

calendar.lotr
Events from Tolkien's "The Lord of the Rings".

calendar.music
Musical events, births, and deaths. Strongly oriented toward rock 'n' roll.

calendar.usholiday
U.S. holidays. This calendar should be updated yearly by the local system administrator so that roving holidays are set correctly for the current year.

NOTES

Older versions of the *calendar* program selected lines which had the correct date anywhere in the line. In this version, the date is only recognized when it occurs first on the line.

SEE ALSO

cpp, *cron*, *mailx*

NAME

cat – concatenate and print files

SYNOPSIS

`cat [-benstuv] [file ...]`

DESCRIPTION

The *cat* utility reads files sequentially, writing them to the standard output. The *file* operands are processed in command line order. A single dash represents the standard input.

The options are as follows:

–b Like **–n** but does not number blank lines.

–e Like **–v** and displays a dollar sign ('$') at the end of each line.

–n Number the output lines beginning at 1.

–s Squeeze multiple adjacent blank lines adjacent empty lines, causing a single blank line to be output.

–t Like **–v** and displays tab characters as 'ˆI' as well.

–u The **–u** option guarantees that the output is unbuffered.

–v Displays non-printing characters so they are visible. Control characters print as 'ˆX' for control-X; the delete character (octal 0177) prints as 'ˆ?'. Non-ascii characters (with the high bit set) are printed as M- (for meta) followed by the character defined by the lower seven bits.

Because of the shell language mechanism used to perform output redirection, the following command will cause the original data in *file1* to be destroyed unless the shell is set for noclobber.

```
cat file1 file 2 > file1
```

DIAGNOSTICS

The *cat* utility exits with 0 on success, and >0 if an error occurs.

SEE ALSO

head, *pr*, *tail*

NAME

cat32 – Win32 program to copy stdin to stdout

SYNOPSIS

```
cat32
```

DESCRIPTION

The *cat32* utility is a Win32 program which copies its standard input to standard output. It has no other options. It is sometimes used when constructing command pipelines from both INTERIX and Win32 commands.

The INTERIX subsystem has some limitations in running Win32 programs and connecting their output to INTERIX pipes. Any access to a particular pipe end by an INTERIX process after a Win32 process has been given that end (via the *exec*( ) call) will fail, either with the [EPIPE] error, a SIGPIPE signal, or EOF.

For example, this command will show only the contents of *file1*:

```
(cmd.exe /c type file1; cat file2) | cat
```

The INTERIX *cat* command fails, because when it tries to write to the pipe (after the Win32 command), it gets an EPIPE error.

This command will work, because a Win32 command can accept the pipe from an INTERIX command:

```
(cat file1 ; cmd.exe /c type file2) | cat
```

The subshell created by the parentheses fork / execs the INTERIX *cat* command first; it's only after it fork / execs the Win32 command that the write end of the pipe becomes inaccessible to INTERIX processes.

The *cat32* utility is a Win32 filter that can overcome many of these problems. The fix for the first command is to use *cat32*:

```
(cmd.exe /c type file1; cat file2 | cat32) | cat
```

In this case, the INTERIX cat is writing to a fresh, uncontaminated pipe.

DIAGNOSTICS

The *cat32* utility exits with status 0 for success, and >0 if an error occurred.

SEE ALSO

cat

exec()

NAME

cd – change directory

SYNOPSIS

```
cd [directory]
cd –
```

DESCRIPTION

The *cd* utility sets the current working directory to *directory*. This is a built-in command of the shell.

If no *directory* is specified, *cd* changes to the *HOME* directory. When the shell successfully changes directories, it stores the name of the old directory in a variable (*OLDPWD* for *ksh*, *owd* in *tcsh*). The command

```
cd –
```

changes back to the previous directory.

If the specified *directory* doesn't begin with a slash (/) and isn't a subdirectory of the current directory, the shell searches through a list of directories for the specified *directory*. In *sh* and *ksh*, the list of directories is stored in the variable *CDPATH*; if the variable is unset or is null, only the current directory is searched. (In the C shell, see the *cdpath* variable.)

SEE ALSO

ksh, *tcsh*

NAME

chgrp – change group

SYNOPSIS

`chgrp [-R] group files ...`

DESCRIPTION

The *chgrp* utility sets the group ID of the file named by each *file* operand to the *group* ID specified by the group operand.

Options:

–R Change the group ID for the file hierarchies rooted in the files instead of just the files themselves.

The *group* operand can be either a group name from the group database, or a numeric group ID. If a group name is also a numeric group ID, the operand is used as a group name.

The user invoking *chgrp* must belong to the specified group and be the owner of the file, or have appropriate privileges.

DIAGNOSTICS

The *chgrp* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

chown

NAME

chmod – change file modes

SYNOPSIS

`chmod [-R] mode file ...`

DESCRIPTION

The *chmod* utility modifies the file mode bits of the listed files as specified by the *mode* operand.

The options are as follows:

–R Change the modes of the file hierarchies rooted in the files instead of just the files themselves.

Only the owner of a file or a user with appropriate privileges is permitted to change the mode of a file.

MODES

Modes may be absolute or symbolic. An absolute mode is an octal number constructed by *or-ing* the following values:

`0400`	read by owner
`0200`	write by owner
`0100`	execute (or search for directories) by owner
`0040`	read by group
`0020`	write by group
`0010`	execute / search by group
`0004`	read by others
`0002`	write by others
`0001`	execute / search by others

The read, write, and execute / search values for group and others are encoded as described for owner.

The symbolic mode is described by the following grammar:

```
mode         ::= clause [, clause ...]
clause       ::= [who ...] [action ...] last_action
action       ::= op [perm ...]
last_action  ::= op [perm ...]
who          ::= a | u | g | o
op           ::= + | - | =
perm         ::= r | w | x | X | u | g | o
```

The *who* symbols (“u”, “g”, and “o”) specify the user, group, and other parts of the mode bits, respectively. The *who* symbol “a” is equivalent to “ugo”.

The *perm* symbols represent the portions of the mode bits as follows:

r	The read bits.
w	The write bits.
x	The execute / search bits.
X	The execute / search bits if the file is a directory or any of the execute / search bits are set in the original (unmodified) mode. Operations with the *perm* symbol "X" are only meaningful in conjunction with the *op* symbol "+", and are ignored in all other cases.
u	User permission bits in the original file's mode.
g	Group permission bits in the original file's mode.
o	other permission bits in the original file's mode.

The *op* symbols represent the operation performed, as follows:

+	If no value is supplied for *perm*, the "+" operation has no effect. If no value is supplied for *who*, each permission bit specified in *perm*, for which the corresponding bit in the file mode creation mask is clear, is set. Otherwise, the mode bits represented by the specified *who* and *perm* values are set.
–	If no value is supplied for *perm*, the "–" operation has no effect. If no value is supplied for *who*, each permission bit specified in *perm*, for which the corresponding bit in the file mode creation mask is clear, is cleared. Otherwise, the mode bits represented by the specified *who* and *perm* values are cleared.
=	The mode bits specified by the *who* value are cleared, or, if no who value is specified, the owner, group and other mode bits are cleared. Then, if no value is supplied for *who*, each permission bit specified in *perm*, for which the corresponding bit in the file mode creation mask is clear, is set. Otherwise, the mode bits represented by the specified *who* and *perm* values are set.

Each *clause* specifies one or more operations to be performed on the mode bits, and each operation is applied to the mode bits in the order specified.

Operations upon the other permissions only (specified by the symbol "o" by itself), in combination with the *perm* symbols "s" or "t", are ignored.

EXAMPLES

The following are example permission modes:

644	Make a file readable by anyone and writeable by the owner only.
go-w	Deny write permission to group and others.
=rw,+X	Set the read and write permissions to the usual defaults, but retain any execute permissions that are currently set.
+X	Make a directory or file searchable / executable by everyone if it is already searchable / executable by anyone.
755 **u=rwx,go=u-w**	Make a file readable / executable by

	everyone and writeable by the owner only.
u=rwx,go=rx	
go=	Clear all mode bits for group and others.
g=u-w	Set the group bits equal to the user bits, but clear the group write bit.

DIAGNOSTICS

The *chmod* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

chgrp, *chown*, *umask*

NAME

chown – change file owner and group

SYNOPSIS

`chown [-R] owner[:group] file ...`

DESCRIPTION

Chown sets the user ID and / or the group ID of the specified files.

The options are as follows:

–R Change the user ID and / or the group ID for the file hierarchies rooted in the files instead of just the files themselves.

The *owner* and *group* operands are both optional, however, one must be specified. If the *group* operand is specified, it must be preceded by a colon (‘‘:’’) character.

The *owner* may be either a numeric user ID or a user name. If a user name is also a numeric user ID, the operand is used as a user name. The *group* may be either a numeric group ID or a group name. If a group name is also a numeric group ID, the operand is used as a group name.

The ownership of a file may only be altered by a process with appropriate privileges for obvious security reasons.

DIAGNOSTICS

The *chown* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

chgrp

NAME

ci – check in RCS revisions

SYNOPSIS

`ci [ options ] file ...`

DESCRIPTION

Ci stores new revisions into RCS files. Each pathname matching an RCS suffix is taken to be an RCS file. All others are assumed to be working files containing new revisions. *ci* deposits the contents of each working file into the corresponding RCS file. If only a working file is given, *ci* tries to find the corresponding RCS file in an RCS subdirectory and then in the working file's directory. For more details, see "File Naming" below.

Ci accepts the following options. Detailed discussion of *ci*'s behavior follows the options.

–d*date* Use *date* for the checkin date and time. The *date* is specified in free format as explained in *co*. This is useful for lying about the checkin date, and for

–f*rev* Force a deposit; the new revision is deposited even it is not different from the preceding one.

–I*rev* Run in interactive mode; the user is prompted and questioned even if the standard input is not a terminal. **–k** if no date is available. If *date* is empty, the working file's time of last modification is used.

–k*rev* Search the working file for keyword values to determine its revision number, creation date, state, and author (see *co*), and assigns these values to the deposited revision, rather than computing them locally. It also generates a default login message noting the login of the caller and the actual checkin date. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the **–k** option at these sites to preserve the original number, date, author, and state. The extracted keyword values and the default log message may be overridden with the options **–d**, **–m**, **–s**, **–w**, and any option that carries a revision number.

–l*[rev]* Like **–r**, except it performs an additional **co –l** for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue editing it after the checkin.

–M*rev* Set the modification time on any new working file to be the date of the retrieved revision. For example, **ci –d –M –u***f* does not alter *f*'s modification time, even if *f*'s contents change due to keyword substitution. Use this option with care; it can confuse *make*.

–m*msg* Use the string *msg* as the log message for all revisions checked in.

–n*name* Assign the symbolic name *name* to the number of the checked-in revision. *ci* prints an error message if *name* is already assigned to another number.

–N*name* Same as **–n**, except that it overrides a previous assignment of *name.*

–q[*rev*] Quiet mode; do not print diagnostic output. A revision that is not different from the preceding one is not deposited, unless **–f** is given.

–r*rev* Check in a revision, releases the corresponding lock, and removes the working file. This is the default.

The **–r** option has an unusual meaning in *ci*. In other RCS commands, **–r** merely specifies a revision number, but in *ci* it also releases a lock and removes the working file. See **–u** for a tricky example.

–s*state* Set the state of the checked-in revision to the identifier *state*. The default state is "Exp".

–t*file* Write descriptive text from the contents of the named *file* into the RCS file, deleting the existing text. The *file* may not begin with **.** (dot).

–t–*string* Write descriptive text from the *string* into the RCS file, deleting the existing text.

The **–t** option, in both its forms, has effect only during an initial checkin; it is silently ignored otherwise.

During the initial checkin, if **–t** is not given, *ci* obtains the text from standard input, terminated by end-of-file or by a line containing **.** by itself. The user is prompted for the text if interaction is possible; see **–I**.

For backward compatibility with older versions of RCS, a bare **–t** option is ignored.

–w*logname*
Use *logname* for the author field of the deposited revision. Useful for lying about the author, and for **–k** if no author is available.

–u[*rev*] Like **–l**, except that the deposited revision is not locked. This lets one read the working file immediately after checkin.

The **–l**, **–r**, and **–u** options are mutually exclusive and silently override each other. For example, **ci –u –r** is equivalent to **ci –R** because **–r** overrides **–u**.

–V*n* Emulate RCS version *n*. See *co* for details.

–x*suffixes*
Identify RCS files by the given *suffixes*. A nonempty suffix matches any pathname ending in the suffix. An empty suffix matches any pathname of the form **RCS/***file* or *path***/RCS/***file*. The **–x** option can specify a list of suffixes separated by / (slash). For example, **–x,v/** specifies two suffixes: **,v** and the empty suffix. If two or more suffixes are specified, they are tried in order when looking for an RCS file; the first one that works is used for that file. If no RCS file is found but an RCS file can be created, the suffixes are tried in order to determine the new RCS file's name. The default for *suffixes* is installation-dependent; normally it is **,v/** on systems that permit commas in file names, and is the empty suffix for other hosts.

For *ci* to work, the caller's login must be on the access list, except if the access list is empty or the caller is the superuser or the owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file if non-strict locking is used (see *rcs*). A lock held by someone else may be broken with the *rcs* command.

Ci checks whether the revision to be deposited differs from the preceding one (unless the **–f** option is given). If the revision doesn't differ, *ci* reverts to the preceding one instead of creating a new revision. To revert, *ci* removes the working file and any lock; this is modified by the options **–l** and **–u**. **ci –l** keeps any lock and **ci –u** removes any lock, and then they both generate a new working file much as if **co –l** or **co –u** had been applied to the preceding revision. When reverting, any **–n** and **–s** options apply to the preceding revision.

For each revision deposited, *ci* prompts for a log message. The log message should summarize the change and must be terminated by end-of-file or by a line containing **.** by itself. If several files are checked in *ci* asks whether to reuse the previous log message. If the standard input is not a terminal, *ci* suppresses the prompt and uses the same log message for all files. See also **–m**.

If the RCS file does not exist, *ci* creates it and deposits the contents of the working file as the initial revision (default number: **1.1**). The access list is initialized to empty. Instead of the log message, *ci* requests descriptive text (see **–t** below).

The number *rev* of the deposited revision can be given by any of the options **–f**, **–I**, **–k**, **–l**, **–M**, **–q**, **–r**, or **–u**. *rev* may be symbolic, numeric, or mixed. If *rev* is $, *ci* determines the revision number from keyword values in the working file.

If *rev* is a revision number, it must be higher than the latest one on the branch to which *rev* belongs, or must start a new branch.

If *rev* is a branch rather than a revision number, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that branch. If *rev* indicates a non-existing branch, that branch is created with the initial revision numbered *rev***.1**.

If *rev* is omitted, *ci* tries to derive the new revision number from the caller's last lock. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a non-tip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are **1**.

If *rev* is omitted and the caller has no lock, but owns the file and locking is not set to *strict*, then the revision is appended to the default branch (normally the trunk; see the **–b** option of *rcs*).

Exception: On the trunk, revisions can be appended to the end, but not inserted.

FILE NAMING

Pairs of RCS files and working files may be specified in three ways (see also the "EXAMPLES" section).

1. Both the RCS file and the working file are given. The RCS pathname is of the form *path1***/***workfileX* and the working pathname is of the form *path2***/***workfile* where *path1***/** and *path2***/** are (possibly different or empty) paths, *workfile* is a filename, and *X* is an RCS suffix. If *X* is empty, *path1***/** must be **RCS/** or must end in **/RCS/**.
2. Only the RCS file is given. Then the working file is created in the current directory and its name is derived from the name of the RCS file by removing *path1***/** and the suffix *X*.

3. Only the working file is given. Then *ci* considers each RCS suffix *X* in turn, looking for an RCS file of the form *path2***/RCS/***workfileX* or (if the former is not found and *X* is nonempty) *path2*/*workfileX*.

If the RCS file is specified without a path in 1) and 2), *ci* looks for the RCS file first in the directory **./RCS** and then in the current directory.

ci reports an error if an attempt to open an RCS file fails for an unusual reason, even if the RCS file's pathname is just one of several possibilities. For example, to suppress use of RCS commands in a directory *d*, create a regular file named *d***/RCS** so that casual attempts to use RCS commands in *d* fail because *d***/RCS** is not a directory.

EXAMPLES

Suppose **,v** is an RCS suffix and the current directory contains a subdirectory **RCS** with an RCS file **io.c,v**. Then each of the following commands check in a copy of **io.c** into **RCS/io.c,v** as the latest revision, removing **io.c**.

```
$ ci io.c
$ ci RCS/io.c,v
$ ci io.c,v
$ ci io.c RCS/io.c,v
$ ci io.c io.c,v
$ ci RCS/io.c,v io.c
$ ci io.c,v io.c
```

Suppose instead that the empty suffix is an RCS suffix and the current directory contains a subdirectory **RCS** with an RCS file **io.c**. Then each of the following commands checks in a new revision.

```
$ ci io.c
$ ci RCS/io.c
$ ci io.c RCS/io.c
$ ci RCS/io.c io.c
```

FILE MODES

An RCS file created by *ci* inherits the read and execute permissions from the working file. If the RCS file exists already, *ci* preserves its read and execute permissions. *ci* always turns off all write permissions of RCS files.

FILES

Several temporary files may be created in the directory containing the working file, and also in the temporary directory (see *TMPDIR* under *ENVIRONMENT*). A semaphore file or files are created in the directory containing the RCS file. With a nonempty suffix, the semaphore names begin with the first character of the suffix; therefore, do not specify a suffix whose first character could be that of a working filename. With an empty suffix, the semaphore names end with **_** so working filenames should not end in **_**.

ci never changes an RCS or working file. Normally, *ci* unlinks the file and creates a new one; but instead of breaking a chain of one or more symbolic links to an RCS file, it unlinks the destination file instead. Therefore, *ci* breaks any hard or symbolic links to any working file it changes; and hard links to RCS files are ineffective, but symbolic links to RCS files are preserved.

The effective user must be able to search and write the directory containing the RCS file. Normally, the real user must be able to read the RCS and working files and to search and write the directory containing the working file; however, some older hosts cannot easily

switch between real and effective users, so on these hosts the effective user is used for all accesses. The effective user is the same as the real user unless your copies of *ci* and *co* have setuid privileges. As described in the next section, these privileges yield extra security if the effective user owns all RCS files and directories, and if only the effective user can write RCS directories.

Users can control access to RCS files by setting the permissions of the directory containing the files; only users with write access to the directory can use RCS commands to change its RCS files. For example, in hosts that allow a user to belong to several groups, one can make a group's RCS directories writable to that group only. This approach suffices for informal projects, but it means that any group member can arbitrarily change the group's RCS files, and can even remove them entirely. Hence more formal projects sometimes distinguish between an RCS administrator, who can change the RCS files at will, and other project members, who can check in new revisions but cannot otherwise change the RCS files.

ENVIRONMENT VARIABLES

The *ci* command makes use of the following environment variables:

RCSINIT

A list of options prepended to the argument list, separated by spaces. A backslash escapes spaces within an option. The *RCSINIT* options are prepended to the argument lists of most RCS commands. Useful *RCSINIT* options include **–q**, **–V**, and **–x**.

TMPDIR

Name of the temporary directory. If not set, the environment variables *TMP* and *TEMP* are inspected instead and the first value found is taken; if none of them are set, a host-dependent default is used, **$OPENNT_ROOT/tmp** on INTERIX systems.

DIAGNOSTICS

For each revision, *ci* prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status is zero if and only if all operations were successful.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 1.12; Release Date: 1998/06/04.
Copyright © 1982, 1988, 1989 by Walter F. Tichy.
Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

co, *ident*, *make*, *rcs*, *rcsdiff*, *rcsmerge*, *rlog*

rcsfile, *rcsintro*

Walter F. Tichy, RCS—A System for Version Control, *Software—Practice & Experience* **15**, 7 (July 1985), 637-654.

NAME

cksum, sum – display file checksums and sizes

SYNOPSIS

```
cksum [-o 1 | 2] [file ...]
sum [-r] [file ...]
```

DESCRIPTION

Both the *cksum* and *sum* utilities calculate the checksum and size of a file. If *cksum* is invoked as *sum*, it behaves as the *sum* utility. The *sum* command is obsolete and should not be used.

The *cksum* utility writes to the standard output three whitespace separated fields for each input file:

```
crc  #_octets file_name
```

These are: a checksum (*crc*); the total number of octets in the file (*#_octets*); and the file name. If no file name is specified, the standard input is used and no file name is written.

The *sum* utility writes to the standard output three whitespace separated fields for each input file:

```
crc #_blocks file_name
```

These are: a checksum (**–crc**); the total number of 512-byte blocks in the file (*#_blocks*); and the file name.

The options for the commands are:

–o *n* Use historic algorithms instead of the default one. *n* can be either 1 or 2:

1 Algorithm 1 is the algorithm used by historic BSD systems as the *sum* algorithm and by historic AT&T System V systems as the *sum* algorithm when using the **–r** option. This is a 16-bit checksum, with a right rotation before each addition; overflow is discarded.

2 Algorithm 2 is the algorithm used by historic AT&T System V systems as the default *sum* algorithm. This is a 32-bit checksum, and is defined as follows:

```
s = sum of all bytes;
r = s % 2^16 + (s % 2^32) / 2^16;
cksum = (r % 2^16) + r / 2^16;
```

Both algorithm 1 and 2 write to the standard output the same fields as the default algorithm except that the size of the file in bytes is replaced with the size of the file in blocks. For historic reasons, the block size is 1024 for algorithm 1 and 512 for algorithm 2. Partial blocks are rounded up.

–r For the *sum* command, this is the equivalent of

```
cksum -o 1
```

Without the **–r** option, the *sum* command is equivalent to

```
cskum -o 2
```

The default CRC used is based on the polynomial used for CRC error checking in the networking standard ISO 8802-3:1989. The CRC checksum encoding is defined by the generating polynomial:

```
G(x) = x^32 + x^26 + x^23 + x^22 + x^16 + x^12 +
     x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1
```

Mathematically, the CRC value corresponding to a given file is defined by the following procedure: The *n* bits to be evaluated are considered to be the coefficients of a mod 2 polynomial M(x) of degree *n–1*. These *n* bits are the bits from the file, with the most significant bit being the most significant bit of the first octet of the file and the last bit being the least significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral number of octets, followed by one or more octets representing the length of the file as a binary value, least significant octet first. The smallest number of octets capable of representing this integer are used.

M(x) is multiplied by x^32 (i.e., shifted left 32 bits) and divided by G(x) using mod 2 division, producing a remainder R(x) of degree <= 31.

The coefficients of R(x) are considered to be a 32-bit sequence.

The bit sequence is complemented and the result is the CRC.

DIAGNOSTICS

The *cksum* utility exits 0 on success, and >0 if an error occurs.

NAME

clear – clear terminal display

SYNOPSIS

`clear`

DESCRIPTION

The *clear* utility clears the screen of your terminal. It leaves the cursor in the "home" position, the top left corner of the screen.

The *clear* command ignores any arguments.

ENVIRONMENT VARIABLES

Clear makes use of these environment variables:

_POSIX_TERM
: When set to "on" this environment variable indicates that POSIX-style terminal semantics are on.

TERM
: Specifies the type of terminal.

TERMCAP
: Contains *either* the location of the **termcap** file, as an absolute pathname (it *must* start with /), or a termcap terminal description entry, to be used instead of the termcap entry for the current terminal type. If it contains an entry, the entry has no newlines.

TERMPATH
: A space- or colon-separated list of absolute filenames. If *TERMCAP* is not set, *clear* searches for the appropriate terminal entry in these files.

NAME

cmp – compare two files

SYNOPSIS

`cmp [-l | -s] file1 file2 [skip1 [skip2]]`

DESCRIPTION

The *cmp* utility compares two *file*s of any type and writes the results to the standard output. By default, *cmp* is silent if the files are the same; if they differ, the byte and line number at which the first difference occurred is reported.

Bytes and lines are numbered beginning with one.

If one of *file1* or *file2* is specified as "-" *cmp* compares the other named file with standard input.

The following options are available:

–l Print the byte number (decimal) and the differing byte values (octal) for each difference.

–s Print nothing for differing files; return exit status only.

The optional arguments *skip1* and *skip2* are the byte offsets from the beginning of *file1* and *file2*, respectively, where the comparison will begin. The offset is decimal by default, but may be expressed as an hexadecimal or octal value by preceding the value with a leading "0x" or "0".

DIAGNOSTICS

The *cmp* utility exits with one of the following values:

0 The files are identical.

1 The files are different; this includes the case where one file is identical to the first part of the other. In the latter case, if the **–s** option has not been specified, *cmp* writes to standard output that EOF was reached in the shorter file (before any differences were found).

>1 An error occurred.

SEE ALSO

comm, *diff*

NAME

co – check out RCS revisions

SYNOPSIS

`co [option] file ....`

DESCRIPTION

co retrieves a revision from each RCS file and stores it into the corresponding working file.

If *file* matches an RCS suffix, it denotes an RCS file; otherwise it denotes a working file. Names are paired as explained in *ci*.

The options for *co* are:

–d*date* Retrieve the latest revision on the selected branch whose checkin date / time is less than or equal to *date*. The date and time may be given in free format. The time zone **LT** stands for local time; other common time zone names are understood. For example, the following *date*s are equivalent if local time is January 11, 1990, 8pm Pacific Standard Time, eight hours west of Coordinated Universal Time (UTC):

8:00 pm lt	
4:00 AM, Jan. 12, 1990	*note: default is UTC*
1990 / 01 / 12 04:00:00	*RCS date format*
Thu Jan 11 20:00:00 1990 LT	*output of* ctime*( ) + LT*
Thu Jan 11 20:00:00 PST 1990	*output of* date
Fri Jan 12 04:00:00 GMT 1990	
Thu, 11 Jan 1990 20:00:00 –0800	
Fri-JST, 1990, 1pm Jan 12	
12-January-1990, 04:00-WET	

Most fields in the date and time may be defaulted. The default time zone is UTC. The other defaults are determined in the order year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the time zone's current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date **20, 10:30** defaults to 10:30:00 UTC of the 20th of the UTC time zone's current month and year. The date / time must be quoted if it contains spaces.

–f[*rev*] Force the overwriting of the working file; useful in connection with **–q**. See also *FILE MODES* below.

–I[*rev*] Set interactive mode; the user is prompted and questioned even if the standard input is not a terminal.

–j*joinlist* Generate a new revision which is the join of the revisions on *joinlist*. This option is largely obsoleted by *rcsmerge* but is retained for backwards compatibility.

The *joinlist* is a comma-separated list of pairs of the form *rev2***:***rev3*, where *rev2* and *rev3* are (symbolic or numeric) revision numbers. For the initial such pair, *rev1* denotes the revision selected by the above options **–f**, . . ., **–w**. For all other

pairs, *rev1* denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, *co* joins revisions *rev1* and *rev3* with respect to *rev2*. This means that all changes that transform *rev2* into *rev1* are applied to a copy of *rev3*. This is particularly useful if *rev1* and *rev3* are the ends of two branches that have *rev2* as a common ancestor. If *rev1*<*rev2*<*rev3* on the same branch, joining generates a new revision which is like *rev3*, but with all changes that lead from *rev1* to *rev2* undone. If changes from *rev2* to *rev1* overlap with changes from *rev2* to *rev3*, *co* reports overlaps as described in *merge*.

For the initial pair, *rev2* may be omitted. The default is the common ancestor. If any of the arguments indicate branches, the latest revisions on those branches are assumed. The options **–l** and **–u** lock or unlock *rev1*.

–kk Generate only keyword names in keyword strings; omit their values. See "KEYWORD SUBSTITUTION" below. For example, for the **Revision** keyword, generate the string **$Revision$** instead of **$Revision: 1.9 $**. This option is useful to ignore differences due to keyword substitution when comparing different revisions of a file.

–kkv Generate keyword strings using the default form, e.g. **$Revision: 1.9 $** for the **Revision** keyword. A locker's name is inserted in the value of the **Header**, **Id**, and **Locker** keyword strings only as a file is being locked, i.e. by **ci –l** and **co –l**. This is the default.

–kkvl Like **–kkv**, except that a locker's name is always inserted if the given revision is currently locked.

–ko Generate the old keyword string, present in the working file just before it was checked in. For example, for the **Revision** keyword, generate the string **$Revision: 1.1 $** instead of **$Revision: 1.9 $** if that is how the string appeared when the file was checked in. This can be useful for binary file formats that cannot tolerate any changes to substrings that happen to take the form of keyword strings.

–kv Generate only keyword values for keyword strings. For example, for the **Revision** keyword, generate the string **1.9** instead of **$Revision: 1.9 $**. This can help generate files in programming languages where it is hard to strip keyword delimiters like **$Revision: $** from a string. However, further keyword substitution cannot be performed once the keyword names are removed, so this option should be used with care. Because of this danger of losing keywords, this option cannot be combined with **–l**, and the owner write permission of the working file is turned off; to edit the file later, check it out again without **–kv**.

–l[*rev*] Same as **–r**, except that it also locks the retrieved revision for the caller.

–M[*rev*] Set the modification time on the new working file to be the date of the retrieved revision. Use this option with care; it can confuse *make*.

–p[*rev*] Print the retrieved revision on the standard output rather than storing it in the working file. This option is useful when *co* is part of a pipe.

–q[*rev*] Quiet mode; diagnostics are not printed.

–r[*rev*] Retrieves the latest revision whose number is less than or equal to *rev*. If *rev* indicates a branch rather than a revision, the latest revision on that branch is retrieved. If *rev* is omitted, the latest revision on the default branch (see the **–b** option of *rcs*) is retrieved. If *rev* is **$**, *co* determines the revision number from keyword values in the working file. Otherwise, a revision is composed of one or more numeric or symbolic fields separated by periods. The numeric equivalent of a symbolic field is specified with the **–n** option of the commands *ci* and *rcs*.

–s*state* Retrieve the latest revision on the selected branch whose state is set to *state*.

–u[*rev*] Same as **–r**, except that it unlocks the retrieved revision if it was locked by the caller. If *rev* is omitted, **–u** retrieve the revision locked by the caller, if there is one; otherwise, it retrieves the latest revision on the default branch.

–V*n* Emulate RCS version *n*, where *n* may be **3**, **4**, or **5**. This may be useful when interchanging RCS files with others who are running older versions of RCS. To see which version of RCS your correspondents are running, have them invoke **rlog** on an RCS file; if none of the first few lines of output contain the string **branch:** it is version 3; if the dates' years have just two digits, it is version 4; otherwise, it is version 5. An RCS file generated while emulating version 3 will lose its default branch. An RCS revision generated while emulating version 4 or earlier will have a timestamp that is off by up to 13 hours. A revision extracted while emulating version 4 or earlier will contain dates of the form *yy/mm/dd* instead of *yyyy/mm/dd* and may also contain different white space in the substitution for **Log**.

–w[*logname*]
Retrieve the latest revision on the selected branch which was checked in by the user with login name *logname*. If the argument *login* is omitted, the caller's login is assumed.

–x*suffixes*
Use *suffixes* to characterize RCS files. See *ci* for details.

Revisions of an RCS file may be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (e.g., compiling) need not be locked. A revision checked out for editing and later checkin must normally be locked. Checkout with locking fails if the revision to be checked out is currently locked by another user. (A lock may be broken with *rcs*.) Checkout with locking also requires the caller to be on the access list of the RCS file, unless he is the owner of the file or the superuser, or the access list is empty. Checkout without locking is not subject to accesslist restrictions, and is not affected by the presence of locks.

A revision is selected by options for revision or branch number, checkin date / time, author, or state. When the selection options are applied in combination, *co* retrieves the latest revision that satisfies all of them. If none of the selection options is specified, *co* retrieves the latest revision on the default branch (normally the trunk, see the **–b** option of *rcs*). A revision or branch number may be attached to any of the options **–f**, **–I**, **–l**, **–M**, **–p**, **–q**, **–r**, or **–u**. The options **–d** (date), **–s** (state), and **–w** (author) retrieve from a single branch, the *selected* branch, which is either specified by one of **–f,** . . ., **–u**, or the default branch.

A *co* command applied to an RCS file with no revisions creates a zero-length working file. *co* always performs keyword substitution (see below).

KEYWORD SUBSTITUTION

Strings of the form **$***keyword***$** and **$***keyword*:...**$** embedded in the text are replaced with strings of the form **$***keyword*:*value***$** where *keyword* and *value* are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings of the form **$***keyword***$**. On checkout, *co* replaces these strings with strings of the form **$***keyword*:*value***$**. If a revision containing strings of the latter form is checked back in, the value fields will be replaced during the next checkout. Thus, the keyword values are automatically updated on checkout. This automatic substitution can be modified by the **–k** options.

Keywords and their corresponding values:

$Author$
The login name of the user who checked in the revision.

$Date$ The date and time (UTC) the revision was checked in.

$Header$
A standard header containing the full pathname of the RCS file, the revision number, the date (UTC), the author, the state, and the locker (if locked).

Id Same as **$Header$**, except that the RCS filename is without a path.

$Locker$
The login name of the user who locked the revision (empty if not locked).

Log The log message supplied during checkin, preceded by a header containing the RCS filename, the revision number, the author, and the date (UTC). Existing log messages are *not* replaced. Instead, the new log message is inserted after **$Log:...$**. This is useful for accumulating a complete change log in a source file.

$RCSfile$
The name of the RCS file without a path.

$Revision$
The revision number assigned to the revision.

$Source$
The full pathname of the RCS file.

$State$ The state assigned to the revision with the **–s** option of *rcs* or *ci*.

FILE MODES

The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on, unless **–kv** is set or the file is checked out unlocked and locking is set to strict (see *rcs*).

If a file with the name of the working file exists already and has write permission, *co* aborts the checkout, asking beforehand if possible. If the existing working file is not writable or **–f** is given, the working file is deleted without asking.

FILES

co accesses files much as *ci* does, except that it does not need to read the working file.

ENVIRONMENT VARIABLES

Co makes use of the following environment variables:

RCSINIT
A list of options prepended to the argument list, separated by spaces. See *ci* for details.

DIAGNOSTICS

The RCS pathname, the working pathname, and the revision number retrieved are written to the diagnostic output. The exit status is zero if and only if all operations were successful.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 1.9; Release Date: 1997 / 11 / 12.
Copyright © 1982, 1988, 1989 by Walter F. Tichy.
Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

ci, *date*, *ident*, *make*, *rcs*, *rcsdiff*, *rcsmerge*, *rlog*

ctime

rcsfile, *rcsintro*

Walter F. Tichy, RCS—A System for Version Control, *Software—Practice & Experience* **15**, 7 (July 1985), 637-654.

LIMITS

Links to the RCS and working files are not preserved.

There is no way to selectively suppress the expansion of keywords, except by writing them differently. In nroff and troff, this is done by embedding the null-character **\&** into the keyword.

BUGS

The **–d** option sometimes gets confused, and accepts no date before 1970.

YEAR 2000 ISSUES

The year of the date is stored as 4 digits, so no year 2000 issues should arise.

NAME

col – filter reverse line feeds from input

SYNOPSIS

`col [ -bfx] [ -l num]`

DESCRIPTION

Col filters out reverse (and half reverse) line feeds so that the output is in the correct order with only forward and half forward line feeds, and replaces white-space characters with tabs where possible. This can be useful in processing the output of *nroff* and *tbl*.

Col reads from the standard input and writes to the standard output.

The options are as follows:

–b Do not output any backspaces, printing only the last character written to each column position.

–f Forward half line feeds are permitted (‘‘fine’’ mode). Normally characters printed on a half line boundary are printed on the following line.

–x Output multiple spaces instead of tabs.

–l*num* Buffer at least *num* lines in memory. By default, 128 lines are buffered.

The control sequences for carriage motion that *col* understands and their decimal values are listed in the following table:

ESC–7	reverse line feed (escape then 7)
ESC–8	half reverse line feed (escape then 8)
ESC–9	half forward line feed (escape then 9)
backspace	moves back one column (8);ignored in the first column
carriage return	(13)
newline	forward line feed (10); also does carriage return
shift in	shift to normal character set (15)
shift out	shift to alternate character set (14)
space	moves forward one column (32)
tab	moves forward to next tab stop (9)
vertical tab	reverse line feed (11)

All unrecognized control characters and escape sequences are discarded.

Col keeps track of the character set as characters are read and makes sure the character set is correct when they are output.

If the input attempts to back up to the last flushed line, *col* will display a warning message.

SEE ALSO

expand, *nroff*, *tbl*

NAME

comm – select or reject lines common to two files

SYNOPSIS

`comm [-123] file1 file2`

DESCRIPTION

The *comm* utility reads *file1* and *file2*, which should be sorted lexically, and produces three text columns as output: lines only in *file1*; lines only in *file2*; and lines in both files.

The filename ‘‘-’’ means the standard input.

The following options are available:

–1 Suppress printing of column 1.

–2 Suppress printing of column 2.

–3 Suppress printing of column 3.

Each column will have a number of tab characters prepended to it equal to the number of lower numbered columns that are being printed. For example, if column number two is being suppressed, lines printed in column number one will not have any tabs preceding them, and lines printed in column number three will have one.

Comm assumes that the files are lexically sorted; all characters participate in line comparisons.

DIAGNOSTICS

Comm exits 0 on success, >0 if an error occurred.

SEE ALSO

cmp, *diff*, *sort*, *uniq*

NAME

command – execute a simple command

SYNOPSIS

command *command_name* **[** *args* **]**

DESCRIPTION

The *command* command is described in the section on Builtin commands in the *sh* reference page.

SEE ALSO

sh

NAME

cp – copy files

SYNOPSIS

```
cp [-fip] source_file target_file
cp [-fip] source_file ... target
cp -R[-fip] source_file ... target
cp -r[-fip] source_file ... target
```

DESCRIPTION

In the first synopsis form, the *cp* utility copies the contents of the *source_file* to the *target_file*. In the second, third, and fourth synopsis form, the contents of each named *source_file* is copied to the destination *target_directory*. The names of the files themselves are not changed. If *cp* detects an attempt to copy a file to itself, the copy will fail.

The following options are available:

–f For each existing destination pathname, remove it and create a new file, without prompting for confirmation regardless of its permissions. (The **–i** option is ignored if the **–f** option is specified.)

–i Inquire before copying a file that would overwrite an existing file. *Cp* writes a prompt to the standard error output; if the response from the standard input begins with the character ‘‘y’’, the file copy is attempted.

–p Preserve in the copy as many of the modification time, access time, file flags, file mode, user ID, and group ID as allowed by permissions.

- If the user ID and group ID cannot be preserved, no error message is displayed and the exit value is not altered.
- If the source file has its set user ID bit on and the user ID cannot be preserved, the set user ID bit is not preserved in the copy’s permissions.
- If the source file has its set group ID bit on and the group ID cannot be preserved, the set group ID bit is not preserved in the copy’s permissions.
- If the source file has both its set user ID and set group ID bits on, and either the user ID or group ID cannot be preserved, neither the set user ID or set group ID bits are preserved in the copy’s permissions.

–R Copy file hierarchies including FIFOs; this may not be specified with the **–r** option.

–r Copy file hierarchies but not including special files such as FIFOs; this may not be specified with the **–R** option.

For each destination file that already exists, its contents are overwritten if permissions allow, but its mode, user ID, and group ID are unchanged.

In the second synopsis form, *target_directory* must exist unless there is only one named *source_file* which is a directory and the **–R** flag is specified.

If the destination file does not exist, the mode of the source file is used as modified by the file mode creation mask.

Appropriate permissions are required for file creation or overwriting.

DIAGNOSTICS

Cp exits 0 on success, >0 if an error occurred.

SEE ALSO

mv, *umask*

NAME

cpio – copy files to and from archives

SYNOPSIS

```
cpio {-o|--create} [-0acvABLV] [-C bytes]
     [-H format] [-M message]
     [-O [[user@]host:]archive]
     [-F [[user@]host:]archive]
     [--file=[[user@]host:]archive]
     [--format=6format] [--message=message]
     [--null] [--reset-access-time] [--verbose]
     [--dot] [--append] [--block-size=blocks]
     [--dereference] [--io-size=bytes] [--quiet]
     [--force-local] [--help] [--version]
     < name-list [> archive]

cpio {-i|--extract} [-bcdfmnrtsuvBSV] [-C bytes]
     [-E file] [-H format] [-M message]
     [-R [user][:.][group]] [-I [[user@]host:]archive]
     [-F [[user@]host:]archive]
     [--file=[[user@]host:]archive]
     [--make-directories] [--nonmatching]
     [--preserve-modification-time] [--numeric-uid-gid]
     [--rename] [--list] [--swap-bytes]
     [--swap] [--dot] [--unconditional] [--verbose]
     [--block-size=blocks] [--swap-halfwords]
     [--io-size=bytes] [--pattern-file=file] [--format=format]
     [--owner=[user][:.][group]] [--no-preserve-owner]
     [--message=message] [--force-local] [--no-absolute-filenames]
     [--sparse] [--only-verify-crc] [--quiet]
     [--help] [--version] [pattern...] [< archive]

cpio {-p|--pass-through} [-0adlmuvV] [-R [user][:.][group]]
     [--null] [--reset-access-time] [--make-directories]
     [--link] [--quiet] [--preserve-modification-time]
     [--unconditional] [--verbose] [--dot] [--dereference]
     [--owner=[user][:.][group]] [--no-preserve-owner]
     [--sparse] [--help] [--version]
     destination-directory < name-list
```

DESCRIPTION

This manual page documents the GNU version of *cpio*. *Cpio* copies files into or out of a cpio or tar archive, which is a file that contains other files plus information about them, such as their file name, owner, timestamps, and access permissions. The archive can be another file on the disk, a magnetic tape, or a pipe. *cpio* has three operating modes.

–o In **copy-out** mode, *cpio* copies files into an archive. It reads a list of filenames, one per line, on the standard input, and writes the archive onto the standard output. A typical way to generate the list of filenames is with the *find* command; you should give *find* the **–depth** option to minimize problems with permissions on directories that are unwritable or not searchable.

–i In **copy-in** mode, *cpio* copies files out of an archive or lists the archive contents. It reads the archive from the standard input. Any non-option command line arguments are shell globbing patterns; only files in the archive whose names match one or more of those patterns are copied from the archive. Unlike in the shell, an initial '.' in a filename does match a wildcard at the start of a pattern, and a '/' in a filename can match wildcards. If no patterns are given, all files are extracted.

–p In **copy-pass** mode, *cpio* copies files from one directory tree to another, combining the copy-out and copy-in steps without actually using an archive. It reads the list of files to copy from the standard input; the directory into which it will copy them is given as a non-option argument.

Cpio supports the following archive formats: binary, old ASCII, new ASCII, crc, HPUX binary, HPUX old ASCII, old tar, and POSIX.1 tar. (See **–H** for the keywords to specify each format.) The binary format is obsolete because it encodes information about the files in a way that is not portable between different machine architectures. The old ASCII format is portable between different machine architectures, but should not be used on file systems with more than 65536 i-nodes. The new ASCII format is portable between different machine architectures and can be used on any size file system, but is not supported by all versions of *cpio*; currently, it is only supported by GNU and System V R4. The crc format is like the new ASCII format, but also contains a checksum for each file which *cpio* calculates when creating an archive and verifies when the file is extracted from the archive. The HPUX formats are provided for compatibility with HPUX's cpio which stores device files differently.

The tar format is provided for compatability with the *tar* program. It can not be used to archive files with names longer than 100 characters, and can not be used to archive "special" (block or character devices) files. The POSIX.1 tar format can not be used to archive files with names longer than 255 characters (less unless they have a "/" in just the right place).

By default, *cpio* creates binary format archives, for compatibility with older *cpio* programs. When extracting from archives, *cpio* automatically recognizes which kind of archive it is reading and can read archives created on machines with a different byte-order.

Some of the options to *cpio* apply only to certain operating modes; see the "SYNOPSIS" section for a list of which options are allowed in which modes.

OPTIONS

–0, **—null**
In copy-out and copy-pass modes, read a list of filenames terminated by a null character instead of a newline, so that files whose names contain newlines can be archived. GNU **find** is one way to produce a list of null-terminated filenames.

–a, **—reset-access-time**
Reset the access times of files after reading them, so that it does not look like they have just been read.

–A, **—append**
Append to an existing archive. Only works in copy-out mode. The archive must be a disk file specified with the **–O** or (**—file**) option.

–b, **—swap**
In copy-in mode, swap both halfwords of words and bytes of halfwords in the data. Equivalent to *–sS*. Use this option to convert 32-bit integers between big-endian and little-endian machines.

–B Set the I/O block size to 5120 bytes. Initially the block size is 512 bytes.

—block-size= *block-size*
Set the I/O block size to *block-size* * 512 bytes.

–c Use the old portable (ASCII) archive format.

–C *io-size*, **—io-size=***io-size*
Set the I/O block size to *io-size* bytes.

–d, **—make-directories**
Create leading directories where needed.

–E *file*, **—pattern-file=***file*
In copy-in mode, read additional patterns specifying filenames to extract or list from *file*. The lines of *file* are treated as if they had been non-option arguments to *cpio*.

–f, **—nonmatching**
Only copy files that do not match any of the given patterns.

–F, **—file=***archive*
Archive filename to use instead of standard input or output.

—force-local
With **–F**, **–I**, or **–O**, take the archive file name to be a local file even if it contains a colon, which would ordinarily indicate a remote host name.

–H *format*, **—format=***format*
Use archive format *format*. The valid formats are listed below; the same names are also recognized in all-caps. The default in copy-in mode is to automatically detect the archive format, and in copy-out mode is "bin".

bin The obsolete binary format.
odc The old (POSIX.1) portable format.
newc The new (SVR4) portable format, which supports file systems having more than 65536 i-nodes.
crc The new (SVR4) portable format with a checksum added.
tar The old tar format.
ustar The POSIX.1 tar format. Also recognizes GNU *tar* archives, which are similar but not identical.
hpbin The obsolete binary format used by HPUX's *cpio* (which stores device files differently).
hpodc The portable format used by HPUX's *cpio* (which stores device files differently).

–i, **—extract**
Run in copy-in mode.

–I *archive*
Archive filename to use instead of standard input.

–k Ignored; for compatibility with other versions of *cpio*.

–l, **—link**
Link files instead of copying them, when possible.

–m, **—preserve-modification-time**
Retain previous file modification times when creating files.

–M *message*, **—message=***message*
Print *message* when the end of a volume of the backup media (such as a tape or a floppy disk) is reached, to prompt the user to insert a new volume. If *message* contains the string "%d", it is replaced by the current volume number (starting at 1).

–n, **—numeric-uid-gid**
In the verbose table of contents listing, show numeric UID and GID instead of translating them into names.

—no-absolute-filenames
In copy-in mode, create all files relative to the current directory, even if they have an absolute file name in the archive.

—no-preserve-owner
In copy-in mode and copy-pass mode, do not change the ownership of the files; leave them owned by the user extracting them. This is the default for non-root users, so that users on System V don't inadvertantly give away files.

–o, **—create**
Run in copy-out mode.

–O *archive*
Archive filename to use instead of standard output.

—only-verify-crc
When reading a CRC format archive in copy-in mode, only verify the CRC's of each file in the archive, don't actually extract the files.

–p, **—pass-through**
Run in copy-pass mode.

—quiet Do not print the number of blocks copied.

–r, **—rename**
Interactively rename files.

–R [*user*][**:.**][*group*], **—owner** [*user*][**:.**][*group*]
In copy-out and copy-pass modes, set the ownership of all files created to the specified user and / or group. Either the user or the group, or both, must be present. If the group is omitted but the ":" or "." separator is given, use the given user's login group. Only the super-user can change files' ownership.

—sparse
In copy-out and copy-pass modes, write files with large blocks of zeros as sparse files.

–s, **—swap-bytes**
In copy-in mode, swap the bytes of each halfword (pair of bytes) in the files.

–S, **—swap-halfwords**
In copy-in mode, swap the halfwords of each word (4 bytes) in the files.

–t, **—list**
Print a table of contents of the input.

–u, **—unconditional**
Replace all files, without asking whether to replace existing newer files with older files.

–v, **—verbose**
List the files processed, or with **–t**, give an ‘ls –l’ style table of contents listing. In a verbose table of contents of a ustar archive, user and group names in the archive that do not exist on the local system are replaced by the names that correspond locally to the numeric UID and GID stored in the archive.

–V, **—dot**
Print a "." for each file processed.

—version
Print the *cpio* program version number and exit.

DIAGNOSTICS

The *cpio* utility exists with status 0 on successful completion, and status >0 if an error occurred.

SEE ALSO

ar, *find*, *ls*, *pax*, *tar*

NAME

cccp, cpp – The GNU C-Compatible Compiler Preprocessor

SYNOPSIS

```
cccp [-$] [-A predicate[(value)]]
     [-C] [-D name[=definition]]
     [-dD] [-dM] [-I directory] [-H]
     [-I-] [-imacros file] [-include file]
     [-idirafter dir] [-iprefix prefix]
     [-iwithprefix dir] [-lang-c]
     [-lang-c++] [-lang-objc] [-lang-objc++]
     [-lint] [-M [-MG]] [-MM [-MG]]
     [-MD file ] [-MMD file ] [-nostdinc]
     [-nostdinc++] [-P] [-pedantic] [-pedantic-errors]
     [-traditional] [-trigraphs] [-U name ]
     [-undef] [-Wtrigraphs] [-Wcomment] [-Wall]
     [-Wtraditional] [infile|-] [outfile|-]
```

DESCRIPTION

The C preprocessor is a *macro processor* that is used automatically by the C compiler to transform your program before actual compilation. It is called a macro processor because it allows you to define *macros*, which are brief abbreviations for longer constructs.

The C preprocessor provides four separate facilities that you can use as you see fit:

- Inclusion of header files. These are files of declarations that can be substituted into your program.
- Macro expansion. You can define *macros*, which are abbreviations for arbitrary fragments of C code, and then the C preprocessor will replace the macros with their definitions throughout the program.
- Conditional compilation. Using special preprocessing directives, you can include or exclude parts of the program according to various conditions.
- Line control. If you use a program to combine or rearrange source files into an intermediate file which is then compiled, you can use line control to inform the compiler of where each source line originally came from.

C preprocessors vary in some details. For a full explanation of the GNU C preprocessor, see the *info* file **cpp.info**, or the manual *The C Preprocessor*. Both of these are built from the same documentation source file, **cpp.texinfo**. The GNU C preprocessor provides a superset of the features of ANSI Standard C.

ANSI Standard C requires the rejection of many harmless constructs commonly used by today's C programs. Such incompatibility would be inconvenient for users, so the GNU C preprocessor is configured to accept these constructs by default. Strictly speaking, to get ANSI Standard C, you must use the options **–trigraphs**, **–undef** and **–pedantic**, but in practice the consequences of having strict ANSI Standard C make it undesirable to do this.

Most often when you use the C preprocessor you will not have to invoke it explicitly: the C compiler will do so automatically. However, the preprocessor is sometimes useful individually.

When you call the preprocessor individually, either name (*cpp* or *cccp*) will do—they are completely synonymous.

The C preprocessor expects two file names as arguments, *infile* and *outfile*. The preprocessor reads *infile* together with any other files it specifies with **#include**. All the output generated by the combined input files is written in *outfile*.

Either *infile* or *outfile* may be **–**, which as *infile* means to read from standard input and as *outfile* means to write to standard output. Also, if *outfile* or both file names are omitted, the standard output and standard input are used for the omitted file names.

OPTIONS

Here is a table of command options accepted by the C preprocessor. These options can also be given when compiling a C program; they are passed along automatically to the preprocessor when it is invoked by the compiler.

–P Inhibit generation of **#**-lines with line-number information in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code and will be sent to a program which might be confused by the **#**-lines.

–C Do not discard comments: pass them through to the output file. Comments appearing in arguments of a macro call will be copied to the output before the expansion of the macro call.

–traditional
Try to imitate the behavior of old-fashioned C, as opposed to ANSI C.

–trigraphs
Process ANSI standard trigraph sequences. These are three-character sequences, all starting with **??**, that are defined by ANSI C to stand for single characters. For example, **??/** stands for \, so **'??/n'** is a character constant for a newline. Strictly speaking, the GNU C preprocessor does not support all programs in ANSI Standard C unless **–trigraphs** is used, but if you ever notice the difference it will be with relief.

You don't want to know any more about trigraphs.

–pedantic
Issue warnings required by the ANSI C standard in certain cases such as when text other than a comment follows **#else** or **#endif**.

–pedantic–errors
Like **–pedantic**, except that errors are produced rather than warnings.

–Wtrigraphs
Warn if any trigraphs are encountered (assuming they are enabled).

–Wcomment, **–Wcomments**
Warn whenever a comment-start sequence **/*** appears in a comment. (Both forms have the same effect).

–Wall Requests both **–Wtrigraphs** and **–Wcomment** (but not **—Wtraditional**).

–Wtraditional
Warn about certain constructs that behave differently in traditional and ANSI C.

–I *directory*

Add the directory *directory* to the end of the list of directories to be searched for header files. This can be used to override a system header file, substituting your own version, since these directories are searched before the system header file directories. If you use more than one **–I** option, the directories are scanned in left-to-right order; the standard system directories come after.

–I–

Any directories specified with **–I** options before the **–I–** option are searched only for the case of **#include** *file*; they are not searched for **#include** <*file*>.

If additional directories are specified with **–I** options after the **–I–**, these directories are searched for all **#include** directives.

In addition, the **–I–** option inhibits the use of the current directory as the first search directory for **#include (dq***file*". Therefore, the current directory is searched only if it is requested explicitly with **–I.** Specifying both **–I–** and **–I.** allows you to control precisely which directories are searched before the current one and which are searched after.

–nostdinc

Do not search the standard system directories for header files. Only the directories you have specified with **–I** options (and the current directory, if appropriate) are searched.

–nostdinc++

Do not search for header files in the C++ specific standard directories, but do still search the other standard directories. (This option is used when building libg++.)

–D *name*

Predefine *name* as a macro, with definition **1**.

–D *name*=*definition*

Predefine *name* as a macro, with definition *definition*. There are no restrictions on the contents of *definition*, but if you are invoking the preprocessor from a shell or shell-like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax. If you use more than one **–D** for the same *name*, the rightmost definition takes effect.

–U *name*

Do not predefine *name*. If both **–U** and **–D** are specified for one name, the **–U** beats the **–D** and the name is not predefined.

–undef Do not predefine any nonstandard macros.

–A *name***(***value***)**

Assert (in the same way as the **#assert** directive) the predicate *name* with tokenlist *value*. Remember to escape or quote the parentheses on shell command lines.

You can use **–A-** to disable all predefined assertions; it also undefines all predefined macros.

–dM Instead of outputting the result of preprocessing, output a list of **#define** directives for all the macros defined during the execution of the preprocessor, including predefined macros. This gives you a way of finding out what is predefined in your version of the preprocessor; assuming you have no file **foo.h**, the command

```
touch foo.h; cpp –dM foo.h
```

will show the values of any predefined macros.

–dD Like **–dM** except in two respects: it does *not* include the predefined macros, and it outputs *both* the **#define** directives and the result of preprocessing. Both kinds of output go to the standard output file.

–M

–M [**–MG**]

Instead of outputting the result of preprocessing, output a rule suitable for *make* describing the dependencies of the main source file. The preprocessor outputs one *make* rule containing the object file name for that source file, a colon, and the names of all the included files. If there are many included files then the rule is split into several lines using \-newline.

–MG says to treat missing header files as generated files and assume they live in the same directory as the source file. It must be specified in addition to **–M**.

This feature is used in automatic updating of makefiles.

–MM [**–MG**]

Like **–M** but mention only the files included with **#include** "*file*". System header files included with **#include** <*file*> are omitted.

–MD *file*

Like **–M**, but the dependency information is written to *file*. This is in addition to compiling the file as specified—**–MD** does not inhibit ordinary compilation the way **–M** does.

When invoking gcc, do not specify the *file* argument. Gcc will create file names made by replacing **.c** with **.d** at the end of the input file names.

In Mach, you can use the utility **md** to merge multiple files into a single dependency file suitable for using with the *make* command.

–MMD *file*

Like **–MD** except mention only user header files, not system header files.

–H Print the name of each header file used, in addition to other normal activities.

–imacros *file*

Process *file* as input, discarding the resulting output, before processing the regular input file. Because the output generated from *file* is discarded, the only effect of **–imacros** *file* is to make the macros defined in *file* available for use in the main input. The preprocessor evaluates any **–D** and **–U** options on the command line before processing **–imacros** *file*.

–include *file*

Process *file* as input, and include all the resulting output, before processing the regular input file.

–idirafter *dir*
Add the directory *dir* to the second include path. The directories on the second include path are searched when a header file is not found in any of the directories in the main include path (the one that **–I** adds to).

–iprefix *prefix*
Specify *prefix* as the prefix for subsequent **–iwithprefix** options.

–iwithprefix *dir*
Add a directory to the second include path. The directory's name is made by concatenating *prefix* and *dir*, where *prefix* was specified previously with **–iprefix**.

–lang-c, **–lang-c++**, **–lang-objc**, **–lang-objc++**
Specify the source language. **–lang-c++** makes the preprocessor handle C++ comment syntax, and includes extra default include directories for C++, and **–lang-objc** enables the Objective C **#import** directive. **–lang-c** explicitly turns off both of these extensions, and **–lang-objc++** enables both.

These options are generated by the compiler driver *gcc*, but not passed from the *gcc* command line.

–lint
Look for commands to the program checker *lint* embedded in comments, and emit them preceded by **#pragma lint**. For example, the comment **/* NOTREACHED */** becomes **#pragma lint NOTREACHED**.

This option is available only when you call *cpp* directly; *gcc* will not pass it from its command line.

–$** Forbid the use of **$ in identifiers. This was formerly required for strict conformance to the C Standard before the standard was corrected.

This option is available only when you call *cpp* directly; *gcc* will not pass it from its command line.

SEE ALSO

Cpp entry in *info*; *The C Preprocessor*, Richard M. Stallman.

gcc; **gcc**(**1**); entry in *info*; *Using and Porting GNU CC* (for version 2.0), Richard M. Stallman.

COPYING

Copyright (c) 1991, 1992, 1993 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

NAME

cron1 – single user cron daemon

SYNOPSIS

`cron1 [-o logfile] [-x debugflags] crontabfile`

DESCRIPTION

The *cron1* utility implements the cron daemon for an individual user of the INTERIX subsystem.

The *cron1* program reads a list of cron directives from the **crontab** file (if it exists), then goes into a loop where it sleeps for a minute, rereads the crontab file if its timestamp has changed since it was last read, then executes any commands listed in the *crontabfile* that are scheduled to execute during the current minute.

The crontab file is not required to exist, and can be deleted and recreated without causing *cron1* to exit. A missing crontab file is equivalent to an empty crontab file. The syntax of the crontab file is described in *crontab*(5).

The *cron1* program writes progress and error messages to stderr. If the **–o** option is supplied, then the messages are instead written to the specified log file. If the **–x** option is supplied, then debug messages are also written, as specified by the *debugflags* argument. The *debugflag* argument is a comma separated list of keywords, each of which enables a different set of debug messages. The following keywords are recognized:

```
ext   sch   proc  pars  load  misc  test  bit   mail
```

Several environment variables are set up automatically by the *cron1* daemon. *OPENNT_ROOT* is set to the directory in which INTERIX is installed. *SYSTEMROOT* is the pathname (in Windows syntax) of the directory in which Windows NT is installed; it is used by some Win32 programs. *TZ* is the time zone; it is used by *date* and a few other INTERIX utilities. *SHELL* is set to **$OPENNT_ROOT/bin/sh**. *PATH* is set to **$OPENNT_ROOT/bin:$OPENNT_ROOT/usr/contrib**. *LOGNAME* and *HOME* are set from the NT user database. All of these variables may be overridden by settings in the crontab, except for *LOGNAME*.

In addition to *LOGNAME*, *HOME*, and *SHELL*, *cron1* will look at *MAILTO* if it has any reason to send mail as a result of running commands in the crontab. If *MAILTO* is defined (and non-empty), mail is sent to the user so named. If *MAILTO* is defined but empty (MAILTO=""), no mail will be sent. Otherwise mail is sent to the owner of the crontab. Mail is sent using *mailx*.

The *cron1* program does not exit unless it is interrupted by a signal.

APPLICATION USAGE

The *cron1* program is normally installed and run as a service by the *cronadmin* program. In this environment, the progress and log messages that *cron1* writes to stderr are sent to **/dev/null** and lost. If your personal cron daemon isn't working and you want to debug it, try typing:

```
cron1 $OPENNT_ROOT/usr/lib/cron/users/$LOGNAME/crontab
```

and watch to see what messages get printed to stderr.

SEE ALSO

cronadmin, *crontab*

crontab(5)

AUTHOR

cron1 is loosely based on the cron daemon written by Paul Vixie <paul@vix.com>

NAME

cronadmin – administer who has INTERIX cron privileges

SYNOPSIS

```
cronadmin add username password
cronadmin grant username
cronadmin help
cronadmin list
cronadmin remove username
cronadmin revoke username
cronadmin start username
cronadmin stop username
cronadmin test username
```

DESCRIPTION

The *cronadmin* utility administers who has privileges to run *cron* jobs. It is used to enable a user to run cron jobs, and to revoke a user's privileges. You need Administrator privileges to run *cronadmin* with any subcommands other than **help** or **list**.

For more information, see the *INTERIX User's Guide*.

Cronadmin has the following subcommands.

add *user password*
: Assign cron privileges to the specified user; you must supply the user's *password*. The effect of this command is to assign the service-logon privilege to the user; install a personal cron daemon for the user as a POSIX service; and start the user's cron daemon. Henceforth, the user's cron daemon will be automatically started each time the system is rebooted.

grant *user*
: Assign the service-logon privileges to the specified *user*. These privileges are automatically granted when the user is added.

list
: List users with *cron* privileges.

remove *user*
: Remove the specified *user*'s *cron* privileges. The effect of this command is to stop and uninstall the user's personal cron daemon. However, the user's service-logon privilege is not revoked; use the revoke subcommand to accomplish this.

revoke *user*
: Revoke the specified *user*'s service-logon privilege.

start *user*
: Start the cron daemon for the specified *user*.

stop *user*
: Stop (kill) the cron daemon for the specified *user*. You can restart the daemon later using the *start* subcommand.

test *user*
: Test the specified *user* for cron privileges. The exit status is 0 if the user has cron privileges, otherwise 1.

In normal use, an administrator runs **cronadmin add** *user* to install a cron daemon for a user. Once a user has a cron daemon, they can schedule jobs using the *crontab* command. The format of the **crontab** file is described in *crontab*.

EXIT STATUS

The *cronadmin* utility returns 0 on success and >0 on failure. It is not a failure if the service does not start.

FILES

The *cronadmin* utility makes use of the following files:

$OPENNT_ROOT/usr/lib/cron/user/$LOGNAME/*

The **crontab** files for specific users. The *LOGNAME* is in the form *domain+username*.

NOTES

The cron daemon, like all services, does not explicitly have permission to work across a network. See the *service* man page for a discussion of this limitation.

SEE ALSO

cron1, *crontab*, *service*

crontab

NAME

crontab – schedule periodic jobs to run in the background

SYNOPSIS

```
crontab [file]
crontab [-e|-l|-r]
```

DESCRIPTION

The *crontab* utility manipulates your crontab entry; your crontab entry is the list of commands and when they should be run in the background. You can replace your crontab entry with a new one specified by *file* (or standard input if you don't specify a *file*), or you can edit your entry directly with the command **crontab –e**.

You cannot run the *crontab* command unless an administrator has given you cron privileges using the *cronadmin* utility.

Crontab takes the following options:

–e Edit your crontab entry; if there's no entry, create an empty one to edit. *Crontab* uses the editor specified by your *EDITOR* environment variable.

–l (The letter ell.) List your crontab entry.

–r Remove your crontab entry.

For the syntax and semantics of a crontab file, see *crontab*.

ENVIRONMENT VARIABLES

Crontab uses the following environment variables:

EDITOR

The name of the editor to be used when editing a crontab entry.

DIAGNOSTICS

On success, *crontab* returns 0; if an error occurred, it returns >0.

SEE ALSO

crontab, *cronadmin*, *cron1*

NAME

csplit – split files based on context information

SYNOPSIS

`csplit [-ks] [-f prefix] [-n number] file arg1 ...argn`

DESCRIPTION

The *csplit* utility breaks the specified *file* into smaller files as directed by the *argn* arguments. The *argn* arguments specify a context for where the file is to be split. If *file* is –, *csplit* reads standard input.

Csplit takes the following options:

–f *prefix* Name the created files *prefix***00**, *prefix***01**, and so on to *prefixnn*. The default prefix is **xx**. If the *prefix* would make the resulting file name longer than {NAME_MAX}, *csplit* exits with a diagnostic message and no files are created.

–k *arg* Leave any previously created files intact if an error occurs. Otherwise, *csplit* removes them.

–n *number*
Use *number* decimal digits to form filenames for the file pieces. The default is 2. If the number of digits, combined with the prefix, would make the resulting file name longer than {NAME_MAX}, *csplit* exits with a diagnostic message and no files are created.

–q Suppress the writing of file size messages to standard output. A synonym for **–s**.

–s Suppress the writing of file size messages to standard output.

The arguments *arg1* through *argn* specify where to split the file. They can be a combination of:

/*rexp*/**[***offset***]**
Create a file using the current line up to but not including the first line that matches the basic regular expression *rexp* plus an optional *offset*. The *offset* is a positive integer preceded by **+** (for a number of lines after the regular expression match) or **-** (for a number of lines before the regular expression match). That boundary, the regular expression match plus the offset (if any), becomes the new current line.

%*rexp*%**[***offset***]**
Skip from the current line up to but not including the first line that matches the basic regular expression *rexp* plus an optional *offset*. This behaves in the same way as /*rexp*/ except that it doesn't create a file.

line_num
Create a file from the current line up to but not including line number *line_num*. The first line is numbered 1, and the line *line_num* becomes the new current line.

{*num*} Repeat previous operand *num* times. If *num* is an asterisk (*), the preceding pattern is repeated until the input is exhausted.

It's always an error if the argument doesn't match; *csplit* fails.

DIAGNOSTICS

If the file is split successfully, *csplit* exits with status 0. Otherwise, it exits with a status

of >0.

SEE ALSO

sed, split

NAME

ctags – create a tags file

SYNOPSIS

`ctags [–BFadtuwvx] [ –f tagsfile] name ...`

DESCRIPTION

Ctags makes a tags file for *ex* from the specified C, Pascal, Fortran, YACC, *lex*, and lisp sources. A tags file gives the locations of specified objects in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and a search pattern for the object definition, separated by white-space. Using the **tags** file, *ex* and *vi* can quickly locate these object definitions (using the **–t** option). Depending upon the options provided to *ctags,* objects will consist of subroutines, **typedef**s, **define**s, **struct**s, **enum**s and **union**s.

–a Append to **tags** file.

–B Use backward searching patterns (?...?).

–d Create tags for **#define**s that don't take arguments; **#defines** that take arguments are tagged automatically.

–F Use forward searching patterns (/.../) (the default).

–f *tagsfile*
Place the tag descriptions in a file called *tagsfile*. The default behaviour is to place them in a file called **tags**.

–t Create tags for **typedef**s, **struct**s, **union**s, and **enum**s.

–u Update the specified files in the **tags** file, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the **tags** file.)

–v Produce on the standard output an index of the form expected by *vgrind* (not supplied with the INTERIX distribution). This listing contains the object name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through *sort*. Sample use:

```
ctags –v files | sort –f > index
vgrind –x index
```

–w Suppress warning diagnostics.

–x Produce on the standard output a list of object names, the line number and file name on which each is defined, as well as the text of that line. This is a simple index which can be printed out as an off-line readable function index.

Files whose names end in **.c** or **.h** are assumed to be C source files and are searched for C-style routine and macro definitions. Files whose names end in **.y** are assumed to be *yacc* source files. Files whose names end in **.l** are assumed to be lisp files if their first non-blank character is ';', '(', or '[', otherwise, they are treated as *lex* files. Other files are first examined to see if they contain any Pascal or Fortran routine definitions, and, if not, are searched for C style definitions.

The tag **main** is treated specially in C programs. The tag formed is created by prepending M to the name of the file, with the trailing **.c** and any leading pathname components removed. This makes use of *ctags* practical in directories with more than one program. This implementation of *ctags* recognizes both the /**/ style of C comments and the C++ //-to-end-of-line comment.

Yacc and *lex* files each have a special tag. **Yyparse** is the start of the second section of the *yacc* file, and **yylex** is the start of the second section of the *lex* file.

FILES

The *ctags* utility makes use of the following files:

tags Default output tags file

DIAGNOSTICS

Ctags exits with a value of 1 if an error occurred, 0 otherwise. Duplicate objects are not considered errors.

NOTES

Recognition of *function*s, *subroutine*s and *procedure*s for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose. *Ctags* doesn't understand about Pascal types.

The method of deciding whether to look for C, Pascal or FORTRAN functions is a hack.

Ctags relies on the input being well formed, and any syntactical errors will completely confuse it. It also finds some legal syntax confusing; for example, since it doesn't understand **#ifdef**s (incidentally, that's a feature, not a bug), any code with unbalanced braces inside **#ifdef**s will cause it to become somewhat disoriented. In a similar fashion, multiple line changes within a definition will cause it to enter the last line of the object, rather than the first, as the searching pattern. The last line of multiple line **typedef**s will similarly be noted.

SEE ALSO

ex, *vi*

NAME

cut – select fields of each line of a file

SYNOPSIS

```
cut –c list [file ...]
cut –f list [–d string] [–s] [file ...]
```

DESCRIPTION

The *cut* utility selects portions of each line (as specified by *list*) from each *file* and writes them to the standard output. If no *file* is specified or if the file is specified as -, the standard input is used. The items specified by *list* can be in terms of column position or in terms of fields delimited by a special character. Column numbering starts from 1.

List is a comma or whitespace separated set of increasing numbers and / or number ranges. Number ranges consist of a number, a dash (–), and a second number and select the fields or columns from the first number to the second, inclusive. Numbers or number ranges may be preceded by a dash, which selects all fields or columns from 1 to the first number. Numbers or number ranges may be followed by a dash, which selects all fields or columns from the last number to the end of the line. Numbers and number ranges may be repeated, overlapping, and in any order. It is not an error to select fields or columns not present in the input line.

The options are as follows:

–c *list* The *list* specifies character positions.

–d *string*
Use the first character of *string* as the field delimiter character instead of the tab character.

–f *list* The *list* specifies fields, delimited in the input by a single tab character. Output fields are separated by a single tab character.

–s Suppresses lines with no field delimiter characters. Unless specified, lines with no delimiters are passed through unmodified.

If *file* is specified on the command line by **–**, standard input is used. If no *file* is specified, standard input is used.

DIAGNOSTICS

Cut exits 0 on success, 1 if an error occurred.

SEE ALSO

paste

NAME

date – display date and time

SYNOPSIS

`date [-u] [+format]`

DESCRIPTION

Date displays the current date and time when invoked without arguments. Providing arguments will format the date and time in a user-defined way.

The options are as follows:

–u Display the date in UCT (universal) time.

+*format*

Display the date and time in the format specified by *format*. *Format* can contain arbitrary text and any of the following conversion specifications:

%A	The full weekday name.
%a	The abbreviated weekday name, where the abbreviation is the first three characters.
%B	The full month name.
%b or %h	The abbreviated month name, where the abbreviation is the first three characters.
%C	The format produced by *asctime*( ): equivalent to %a %b %e %H:%M%S %Y .
%c	Equivalent to %m / %d / %y.
%D	The date in the format mm / dd / yy.
%d	The day of the month as a decimal number (01–31).
%e	The day of month as a decimal number (1–31); single digits are preceded by a blank.
%H	The hour (24-hour clock) as a decimal number (00–23).
%I	The hour (12-hour clock) as a decimal number (01–12).
%j	The day of the year as a decimal number (001–366).
%k	The hour (24-hour clock) as a decimal number (0–23); single digits are preceded by a blank.
%l	The hour (12-hour clock) as a decimal number (1–12); single digits are preceded by a blank.
%M	The minute as a decimal number (00–59).
%m	The month as a decimal number (01–12).
%n	A newline.
%p	Either "AM" or "PM" as appropriate.
%R	Equivalent to %H:%M
%r	Equivalent to %I:%M:%S%p.
%t	A tab character.
%S	The second as a decimal number (00–60).
%T or %X	Equivalent to %H:%M:%S.
%U	The week number of the year as a decimal number (00–53), where Sunday is the first day of the week.
%W	The week number of the year as a decimal number (00–53), where Monday is the first day of the week..
%w	The weekday as a decimal number (0–6), where Sunday is the first day of the week.

%x	Equivalent to %m/%d/%y%H:%M:%S.
%Y	The year with century as a decimal number.
%y	The year without century as a decimal number (00–99).
%Z	The time zone name.
%%	Prints a percent character (%).

The format string for the default display is:

```
%a %b %e %H:%M:%S %Z n
```

EXAMPLES

The command:

```
date "+DATE: %m/%d/%y%nTIME: %H:%M:0n"
```

will display:

```
DATE: 11/21/87
TIME: 13:36:16
```

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *date*:

TZ The timezone to use when displaying dates.

DIAGNOSTICS

The *date* command exits with these values:

0 Success

>0 An error occurred.

YEAR 2000 ISSUES

Be careful of the following issues regarding the changeover from 1999 to 2000.

Note that the specifiers %c, %D, %x, and %y all represent the year as a two-digit variable. Examine any shell scripts which make use of this output to ensure that they will function correctly.

SEE ALSO

ctime(), *printf*()

NAME

dc – an arbitrary precision calculator

SYNOPSIS

dc

DESCRIPTION

Dc is a reverse-polish desk calculator which supports unlimited precision arithmetic. It also allows you to define and call macros. Normally *dc* reads from the standard input; if any command arguments are given to it, they are filenames, and *dc* reads and executes the contents of the files before reading from standard input. All normal output is to standard output; all error output is to standard error.

A reverse-polish calculator stores numbers on a stack. Entering a number pushes it on the stack. Arithmetic operations pop arguments off the stack and push the results.

To enter a number in *dc*, type the digits with an optional decimal point. Exponential notation is not supported. To enter a negative number, begin the number with ‘‘_’’. ‘‘-’’ cannot be used for this, as it is a binary operator for subtraction instead. To enter two numbers in succession, separate them with spaces or newlines. These have no meaning as commands.

Printing Commands

p Prints the value on the top of the stack, without altering the stack. A newline is printed after the value.

P Prints the value on the top of the stack, popping it off, and does not print a newline after.

f Prints the entire contents of the stack without altering anything. This is a good command to use if you are lost or want to figure out what the effect of some command has been.

Arithmetic

+ Pops two values off the stack, adds them, and pushes the result. The precision of the result is determined only by the values of the arguments, and is enough to be exact.

– Pops two values, subtracts the first one popped from the second one popped, and pushes the result.

***** Pops two values, multiplies them, and pushes the result. The number of fraction digits in the result is controlled by the current precision value (see below) and does not depend on the values being multiplied.

/ Pops two values, divides the second one popped from the first one popped, and pushes the result. The number of fraction digits is specified by the precision value.

% Pops two values, computes the remainder of the division that the **/** command would do, and pushes that. The division is done with as many fraction digits as the precision value specifies, and the remainder is also computed with that many fraction digits.

^ Pops two values and exponentiates, using the first value popped as the exponent and the second popped as the base. The fraction part of the exponent is ignored. The precision value specifies the number of fraction digits in the result.

v Pops one value, computes its square root, and pushes that. The precision value specifies the number of fraction digits in the result.

Most arithmetic operations are affected by the precision value", which you can set with the **k** command. The default precision value is zero, which means that all arithmetic except for addition and subtraction produces integer results.

The remainder operation **%** requires some explanation: applied to arguments ''a'' and ''b'' it produces ''a - (b * (a / b))'', where ''a / b'' is computed in the current precision.

Stack Control

c Clears the stack, rendering it empty.

d Duplicates the value on the top of the stack, pushing another copy of it. Thus, ''4d*p'' computes 4 squared and prints it.

Registers

Dc provides 256 memory registers, each named by a single character. You can store a number or a string in a register and retrieve it later.

s*r* Pop the value off the top of the stack and store it into register *r*.

l*r* Copy the value in register *r* and push it onto the stack. This does not alter the contents of *r*.

Each register also contains its own stack. The current register value is the top of the register's stack.

S*r* Pop the value off the top of the (main) stack and push it onto the stack of register *r*. The previous value of the register becomes inaccessible.

L*r* Pop the value off the top of register *r*'s stack and push it onto the main stack. The previous value in register *r*'s stack, if any, is now accessible via the **l***r* command.

Parameters

Dc has three parameters that control its operation: the precision, the input radix, and the output radix. The precision specifies the number of fraction digits to keep in the result of most arithmetic operations. The input radix controls the interpretation of numbers typed in; all numbers typed in use this radix. The output radix is used for printing numbers.

The input and output radices are separate parameters; you can make them unequal, which can be useful or confusing. The input radix must be between 2 and 36 inclusive. The output radix must be at least 2. The precision must be zero or greater. The precision is always measured in decimal digits, regardless of the current input or output radix.

i Pops the value off the top of the stack and uses it to set the input radix.

o Pops the value off the top of the stack and uses it to set the output radix.

k Pops the value off the top of the stack and uses it to set the precision.

I Pushes the current input radix on the stack.

O Pushes the current output radix on the stack.

K Pushes the current precision on the stack.

Strings

Dc can operate on strings as well as on numbers. The only things you can do with strings are print them and execute them as macros (which means that the contents of the string are processed as *dc* commands). All registers and the stack can hold strings, and *dc* always knows whether any given object is a string or a number. Some commands such as arithmetic operations demand numbers as arguments and print errors if given strings. Other commands can accept either a number or a string; for example, the **p** command can accept either and prints the object according to its type.

[*characters***]**
Makes a string containing *characters* (contained between balanced **[** and **]** characters), and pushes it on the stack. For example, **[foo]P** prints the characters **foo** (with no newline).

x Pops a value off the stack and executes it as a macro. Normally it should be a string; if it is a number, it is simply pushed back onto the stack. For example, **[1p]x** executes the macro **1p** which pushes **1** on the stack and prints **1** on a separate line.

Macros are most often stored in registers; **[1p]sa** stores a macro to print **1** into register **a**, and **lax** invokes this macro.

>*r* Pops two values off the stack and compares them assuming they are numbers, executing the contents of register *r* as a macro if the original top-of-stack is greater. Thus, **1**2>a will invoke register **a**'s contents and **2**1>a will not.

<*r* Similar but invokes the macro if the original top-of-stack is less.

=*r* Similar but invokes the macro if the two numbers popped are equal.

? Reads a line from the terminal and executes it. This command allows a macro to request input from the user.

q exits from a macro and also from the macro which invoked it. If called from the top level, or from a macro which was called directly from the top level, the **q** command will cause *dc* to exit.

Q Pops a value off the stack and uses it as a count of levels of macro execution to be exited. Thus, **3Q** exits three levels. The **Q** command will never cause *dc* to exit.

Status Inquiry

Z Pops a value off the stack, calculates the number of digits it has (or number of characters, if it is a string) and pushes that number.

X Pops a value off the stack, calculates the number of fraction digits it has, and pushes that number. For a string, the value pushed is 0.

z Pushes the current stack depth; the number of objects on the stack before the execution of the **z** command.

Miscellaneous

! Will run the rest of the line as a system command.

Will interpret the rest of the line as a comment.

:*r* Will pop the top two values off of the stack. The old second-to-top value will be stored in the array *r*, indexed by the old top-of-stack value.

;*r* Pops the top-of-stack and uses it as an index into the array *r*. The selected value is then pushed onto the stack.

NOTES

The array operations **:** and **;** are usually only used by traditional implementations of *bc*. (The GNU *bc* is self-contained and does not need *dc* to run.) The comment operator **#** is a new command not found in traditional implementations of *dc*.

BUGS

Email bug reports to **bug-gnu-utils@prep.ai.mit.edu**. Be sure to include the word ‘‘dc’’ somewhere in the ‘‘Subject:’’ field.

NAME

dd – convert and copy a file

SYNOPSIS

`dd [operands ...]`

DESCRIPTION

The *dd* utility copies the standard input to the standard output. Input data is read and written in 512-byte blocks. If input reads are short, input from multiple reads are aggregated to form the output block. When finished, *dd* displays the number of complete and partial input and output blocks and truncated input records to the standard error output.

The following operands are available:

bs=*n* Set both input and output block size, superseding the **ibs** and **obs** operands. If no conversion values other than **noerror**, **notrunc** or **sync** are specified, then each input block is copied to the output as a single block without any aggregation of short blocks.

cbs=*n* Set the conversion record size to *n* bytes. The conversion record size is required by the record oriented conversion values.

count=*n*
Copy only *n* input blocks.

files=*n* Copy *n* input files before terminating. This operand is only applicable when the input device is a tape.

ibs=*n* Set the input block size to *n* bytes instead of the default 512.

if=*file* Read input from *file* instead of the standard input.

obs=*n* Set the output block size to *n* bytes instead of the default 512.

of=*file* Write output to *file* instead of the standard output. Any regular output file is truncated unless the **notrunc** conversion value is specified. If an initial portion of the output file is skipped (see the **seek** operand) the output file is truncated at that point.

seek=*n* Seek *n* blocks from the beginning of the output before copying. On non-tape devices, a **lseek** operation is used. If the seek operation is past the end of file, space from the current end of file to the specified offset is filled with blocks of NUL bytes.

skip=*n* Skip *n* blocks from the beginning of the input before copying. On input which supports seeks, a **lseek** operation is used. Otherwise, input data is read and discarded. For pipes, the correct number of bytes is read. For all other devices, the correct number of blocks is read without distinguishing between a partial or complete block being read.

conv=*value*[**,** *value* ...]
Where *value* is one of the symbols from the following list.

ascii, **oldascii**
The same as the **unblock** value except that characters are translated from EBCDIC to ASCII before the records are converted. (These values imply **unblock** if the operand **cbs** is also specified.) There are two conversion maps for ASCII. The value **ascii** specifies the recommended one which is compatible with System V. The value **oldascii** specifies the one used in historic AT&T and pre-4.3BSD-reno systems.

block Treats the input as a sequence of newline or end-of-file terminated variable length records independent of input and output block boundaries. Any trailing newline character is discarded. Each input record is converted to a fixed length output record where the length is specified by the **cbs** operand. Input records shorter than the conversion record size are padded with spaces. Input records longer than the conversion record size are truncated. The number of truncated input records, if any, are reported to the standard error output at the completion of the copy.

ebcdic, **ibm**, **oldebcdic**, **oldibm**
The same as the **block** value except that characters are translated from ASCII to EBCDIC after the records are converted. (These values imply **block** if the operand **cbs** is also specified.) There are four conversion maps for EBCDIC. The value **ebcdic** specifies the recommended one which is compatible with AT&T System V. The value **ibm** is a slightly different mapping, which is compatible with the AT&T System V **ibm** value. The values **oldebcdic** and **oldibm** are maps used in historic AT&T and pre-4.3BSD-reno systems.

lcase Transform uppercase characters into lowercase characters.

noerror Do not stop processing on an input error. When an input error occurs, a diagnostic message followed by the current input and output block counts will be written to the standard error output in the same format as the standard completion message. If the **sync** conversion is also specified, any missing input data will be replaced with NUL bytes (or with spaces if a block oriented conversion value was specified) and processed as a normal input buffer. If the **sync** conversion is not specified, the input block is omitted from the output. On input files which are not tapes or pipes, the file offset will be positioned past the block in which the error occurred using **lseek**.

notrunc Do not truncate the output file. This will preserve any blocks in the output file not explicitly written by *dd*. The **notrunc** value is not supported for tapes.

osync Pad the final output block to the full output block size. If the input file is not a multiple of the output block size after conversion, this conversion forces the final output block to be the same size as preceding blocks for use on devices that require regularly sized blocks to be written. This option is incompatible with use of the **bs=***n* block size specification.

swab Swap every pair of input bytes. If an input buffer has an odd number of bytes, the last byte will be ignored during swapping.

sync Pad every input block to the input buffer size. Spaces are used for pad bytes if a block oriented conversion value is specified, otherwise NUL bytes are used.

ucase Transform lowercase characters into uppercase characters.

unblock
Treats the input as a sequence of fixed length records independent of input and output block boundaries. The length of the input records is specified by the **cbs** operand. Any trailing space characters are discarded and a newline character is appended.

Where sizes are specified, a decimal number of bytes is expected. If the number ends with a ‘‘b’’, ‘‘k’’, ‘‘m’’ or ‘‘w’’, the number is multiplied by 512, 1024 (1K), 1048576 (1M)

or the number of bytes in an integer, respectively. Two or more numbers may be separated by an ‘‘x’’ to indicate a product.

When finished, *dd* displays the number of complete and partial input and output blocks, truncated input records and odd-length byte-swapping blocks to the standard error output. A partial input block is one where less than the input block size was read. A partial output block is one where less than the output block size was written. Partial output blocks to tape devices are considered fatal errors. Otherwise, the rest of the block will be written. Partial output blocks to character devices will produce a warning message. A truncated input block is one where a variable length record oriented conversion value was specified and the input line was too long to fit in the conversion record or was not newline terminated.

Normally, data resulting from input or conversion or both are aggregated into output blocks of the specified size. After the end of input is reached, any remaining output is written as a block. This means that the final output block may be shorter than the output block size.

If *dd* receives a SIGINT signal, the current input and output block counts will be written to the standard error output in the same format as the standard completion message and *dd* will exit.

EXAMPLES

To copy from tape drive 0 to tape drive 1:

```
dd if=/dev/tape0 of=/dev/tape1
```

To read an EBCDIC tape, with blocks of ten 80-byte EBCDIC card images per block into the file **cards**:

```
dd if=/dev/tape0 of=cards ibs=800 cbs=80 conv=ascii,lcase
```

Dd is normally used for converting data from one medium (such as tape) to another, but it can also be used for other conversions. This command strips the first 128 bytes from a file:

```
dd bs=1 skip=128 if=infile of=outfile
```

DIAGNOSTICS

The *dd* utility exits 0 on success and >0 if an error occurred.

SEE ALSO

cp, *mt*, *sed*, *tr*

NAME

df – display free disk space

SYNOPSIS

`df [-akP] [-T type] [file | filesystem ...]`

`df -a`

DESCRIPTION

Df displays statistics about the amount of free disk space on the specified *filesystem* or on the filesystem of which *file* is a part. Values are displayed in 512-byte per block block counts. If neither a file or a filesystem operand is specified, *df* displays statistics for drives C through Z. (*df* reports on drives //A and //B if they are specified as operands, however.)

The following options are available:

–a Report statistics on all drives, **//A** through **//Z**.

–k Blocks are in counts of 1024 bytes (one kilobyte) instead of 512 bytes.

–P Prevent the display of the "Type" column in the output. This is the portable output format specified in the POSIX.2 standard.

–T *type* Print only statistics for filesystems of the specified *type*. The recognized *type*s are:

all	All filesystem types.
cdfs	CD-ROM filesystem
fat	File Allocation Table filesystem (such as on DOS partitions).
hpfs	IBM's High Performance File system.
ntfs	New Technology File System.
ofs	Object File System.
samba	Networked SAMBA filesystem.

Typical output is:

```
$ df
Filesystem             512-blocks     Used Available Capacity Type  Mounted on

//Harddisk0/Partition1    3169088  1691069   1478019    53%   ntfs  //C/
//CdRom0                  1085320  1085320         0   100%   cdfs  //D/
//MORGAN/c_drive          2491713  2308884    182829    93%   ntfs  //M/
//MORGAN/nt_src            792272   748651     43621    94%   ntfs  //N/
//MORGAN/products          979760   526281    453479    54%   ntfs  //P/
//MORGAN/VSC_suites       2491713  2308884    182829    93%   ntfs  //W/

$ df -kP //D
Filesystem  1024-blocks     Used Available Capacity Mounted on
//CdRom0         542660   542660         0   100%   //D/
```

The columns are:

Filesystem
: The pathname of the filesystem.

512-blocks

Number of blocks in the filesystem. With the **–k** option, these blocks are 1024-bytes, not 512.

Used Number of blocks currently in use.

Available
Number of blocks available.

Capacity
Current usage expressed as a percentage.

Type The type of the filesystem. This can be any of the values acceptable as arguments to the **–T** option, or **???** if the filesystem type is unknown.

Mounted
The filesystem's mount point.

DIAGNOSTICS

The *df* utility exits with status 0 on success, >0 otherwise.

SEE ALSO

du, *find*

NAME

diff – display file or directory differences

SYNOPSIS

`diff [-cefn] [-bhiwt] file1 file2`

`diff [-D string] [-biw] file1 file2`

`diff [-l] [-r] [-s] [-cefn] [-bhiwt] [-S name] dir1 dir2`

DESCRIPTION

The *diff* utility compares the contents of *file1* and *file2* and writes to the standard output the list of changes necessary to convert one file into the other. No output is produced if the files are identical.

Output options; most of these are mutually exclusive, although some simply modify others:

–*n* Display *n* lines of context. This option is only valid if an unadorned **–c** or **–u** option is also given in the command line.

–C *n* Display *n* lines of context. The **–C** option requires an argument; otherwise it is identical to **–c**.

–c[*n*] Display *n* lines of context, e.g., **–c10** presents 10 lines of context. The default is to present 3 lines of context. With **–c** the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with fifteen *'s. The lines removed from *file1* are marked with '– '; those added to *file2* are marked '+ '. Lines which are changed from one file to the other are marked in both files with '! '. Changes which lie within <context> lines of each other are grouped together on output. (This is a change from the previous ''diff -c'' but the resulting output is usually much easier to interpret.)

Other options affecting this output format are **–F**, **–L**, and **–P**.

–D *string*

Create a merged version of *file1* and *file2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *file1*, while defining *string* will yield *file2*.

–e Produce output in a form suitable as input for the editor utility, *ed*, which can then be used to convert file1 into file2.

Extra commands are added to the output when comparing directories with **–e**, so that the result is a *sh* script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2*.

–f Like **–e**, but the output is in reverse order. It cannot be digested by *ed*.

–h Use an alternate algorithm which can handle files of very long lengths. There is a trade off; the algorithm can only deal with changes which are clearly delimited and brief. Long sections of changes and overlaps will confuse it.

––left-column Output only the left column of common lines. This option is only valid if **–y** (side-by-side display) is specified.

–n Produce a script similar to that of **–e**, but in the opposite order and with a count of changed lines on each insert or delete command. This is the form used by *rcsdiff*.

––suppress-common-lines Do not display common lines. This option is only valid if **–y** (side-by-side display) is specified.

–U *n* Show *n* lines of context, but merge those context lines into a single display rather than copying them. The differing lines are identified by prefixed characters. Requires an argument but otherwise identical to **–u**.

–u[*n*] Show *n* lines of context, but merge those context lines into a single display, rather than copying them. The differing lines are identified by prefixed characters. The default is 3 lines of context. Other options affecting this output format are **–F**, **–L**, and **–P**.

–w *n* This option is only useful if the **–y** option is specified. The maximum length ofor each column is *n* characters per line (default is 130).

–y Write side-by-side output. Additional options to modify this format are **–w**, **––left-column**, and **––suppress-common-lines**.

Comparison options:

–b Ignore trailing blanks (spaces and tabs) and other strings of blanks to compare equal.

–i ignore case of letters, so “A” compares equal to “a”.

–t Expand tabs in output lines. Normal or **–c** output adds character(s) to the front of each line which may screw up the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source’s indentation.

–w Like **–b** but totally ignore whitespace (blanks and tabs). E.g., ‘‘if (a == b)’’ will compare equal to ‘‘if(a==b)’’.

Directory comparison options:

–l Use long output format; each text file *diff*´d is piped through *pr* to paginate it, other differences are remembered and summarized after all text file differences are reported.

–r Apply *diff* recursively to common subdirectories encountered.

–s Report files which are the same; otherwise, these are not mentioned.

–S *name* Re-start a directory *diff* in the middle beginning with file *name.*

If both arguments are directories, *diff* sorts the contents of the directories by name, and then runs the regular file *diff* algorithm, producing a change list, on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are described as such.

If only one of *file1* and *file2* is a directory, *diff* is applied to the non-directory file and the file contained in the directory file with a filename that is the same as the last component of the non-directory file.

If either *file1* or *file2* is **–**, the standard input is used in its place.

Output

The default (without **–e**, **–c**, or **–n** options) output contains lines of these forms, where *XX*, *YY*, *ZZ*, and *QQ* are line numbers respective of file order.

*XX***a***YY* At (the end of) line *XX* of *file1*, append the contents of line *YY* of *file2* to make them equal.

*XX***a***YY***,***ZZ*
Same as above, but append the range of lines, *YY* through *ZZ* of *file2* to line *XX* of file1.

*XX***d***YY* At line *XX* delete the line. The value *YY* tells to which line the change would bring *file1* in line with *file1*.

*XX***,***YY***d***ZZ*
Delete the range of lines *XX* through *YY* in *file1*.

*XX***c***YY* Change the line *XX* in *file1* to the line *YY* in *file2*.

*XX***,***YY***c***ZZ*
Replace the range of specified lines with the line *ZZ*.

*XX***,***YY***c**,*QQ*
Replace the range *XX*,*YY* from *file1* with the range *ZZ*,*QQ* from *file2*.

These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the action letters pertain to *file1*; those after pertain to *file2*. Thus, by exchanging **a** for **d** and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in *ed*, identical pairs (where num1 = num2) are abbreviated as a single number.

ENVIRONMENT VARIABLES

Diff makes use of the following environment variables, if set:

TMPDIR
If the environment variable *TMPDIR* exists, *diff* will use the directory specified by *TMPDIR* as the temporary directory.

FILES

Diff makes use of the following files:

$OPENNT_ROOT/tmp/d?????
Temporary files.

$OPENNT_ROOT/bin/diff
Used for directory diffs.

$OPENNT_ROOT/bin/pr
used by the **–l** option.

SEE ALSO

cmp, *cc*, *comm*, *ed*, *diff3*

DIAGNOSTICS

The *diff* utility exits with one of the following values:

0 No differences were found.
1 Differences were found.
>1 An error occurred.

BUGS

The **–f** and **–e** options do not provide special handling for lines on which the first and only character is . This can cause problems for *ed*.

When comparing directories with the **–b**, **–w**, or **–i** options specified, *diff* first compares the files in the manner of *cmp*, and then decides to run the *diff* algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant white space or case differences.

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

`diff3 [-exEX3] file1 file2 file3`

DESCRIPTION

The *diff3* utility compares the contents of three different versions of a file, *file1*, *file2* and *file3*, writing the result to the standard output. The options describe different methods of merging and purging the separate versions into a new file. *Diff3* is used by the RCS commands to merge specific versions or create new versions.

Options are:

–e Produce output in a form suitable as an input script for the *ed* utility. The script may then be used to merge differences common between all three files and differences specific to *file1* and *file3*. In other words, the **–e** option ignores differences specific to *file1* and *file2*, and those specific to *file2* and *file3*. It is useful for backing out changes specific to *file2* only.

–x Produce an output script suitable for *ed* with changes specific only to all three versions.

–3 Produce an output script suitable for *ed* with changes specific only to *file3*.

–E Similar to **–e** but treat overlapping changes (i.e., changes that would be noted with ==== in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by "<<<<<<" and ">>>>>>" lines.

–X Similar to **–x,** but treat overlapping changes (as described for **–E**) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by "<<<<<<" and ">>>>>>" lines.

The **–E** option is used by RCS *merge* to insure that overlapping changes in the merged files are preserved and brought to someone's attention.

For example, suppose lines 7-8 are changed in both *file1* and *file2*. Applying the edit script generated by the command

```
diff3 -E file1 file2 file3
```

to *file1* results in the file:

```
lines 1-6
of file1
<<<<<<< file1
lines 7-8
of file1
=======
lines 7-8
of file3
>>>>>>> file3
rest of file1
```

The default output of *diff3* makes notation of the differences between all files, and those differences specific to each pair of files. The changes are described by the commands

necessary for *ed* to create the desired target from the different versions. See *diff* for a description of the commands.

==== The lines beneath this notation are ranges of lines which are different between all files.

====*n* The lines beneath this notation are ranges of lines which are exclusively different in file *n*.

FILES

Diff3 makes use of the following files:

$OPENNT_ROOT/tmp/d3*?????*
Temporary files.

$OPENNT_ROOT/bin/diff3
The executable. The root of the path depends upon where the INTERIX distribution is installed.

BUGS

The **–e** option cannot catch and change lines which have . as the first and only character on the line. The resulting script will fail on that line as . is an *ed* editing command.

SEE ALSO

diff, *ed*, *rcs*

NAME

dircmp – compare directory contents

SYNOPSIS

`dircmp [-ds] dir1 dir2`

DESCRIPTION

The *dircmp* utility compares the contents of the directory hierarchies specified by *dir1* and *dir2*. The files are compared by name first and then by content. Files which have identical content but different names will never be compared; use *cmp* or *diff* for that.

The *dircmp* utility takes the following options:

–d Run *diff* on files which have the same name in both directories.

–s Suppress messages about identical files.

See the *diff* reference page for more information on output formats.

DIAGNOSTICS

The *dircmp* utility exits with a value of 0 for success or >0 if an error occurred.

NOTES

This utility is identical to *diff* when used to compare directories.

SEE ALSO

cmp, *diff*

NAME

dirname – return directory portion of pathname

SYNOPSIS

```
dirname string
```

DESCRIPTION

Dirname deletes the filename portion of *string*, beginning with the last slash / character to the end of *string*, and writes the result to the standard output.

DIAGNOSTICS

The *dirname* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

sh

NAME

du – display disk usage statistics

SYNOPSIS

`du [-a|-s] [-H|-L|-P] [-ckx] [`*`file ...`*`]`

DESCRIPTION

The *du* utility displays the file system block usage for each file argument and for each directory in the file hierarchy rooted in each directory argument. If no file is specified, the block usage of the hierarchy rooted in the current directory is displayed. The number of blocks are in the same units as that returned by the *stat*( ) call, i.e. 512-byte blocks, unless the **–k** option is given. Partial numbers of blocks are rounded up.

The options are as follows:

–a Display an entry for each file in the file hierarchy.

–c Force the display of a cumulative total, regardless of other options.

–H Follow symbolic links on the command line. Symbolic links encountered in the tree traversal are not followed. (Symbolic links are not supported in this version of INTERIX.)

–k Use a blocksize of 1024-bytes.

–L All symbolic links are followed. (Symbolic links are not supported in this version of INTERIX.)

–P No symbolic links are followed. (Symbolic links are not supported in this version of INTERIX.)

–s Display only the grand total for the specified files.

–x Filesystem mount points are not traversed.

Du counts the storage used by symbolic links and not the files they reference unless the **–H** or **–L** option is specified. If either the **–H** or **–L** options are specified, storage used by any symbolic links which are followed is not counted or displayed. The **–H**, **–L** and **–P** options override each other and the command's actions are determined by the last one specified.

Files having multiple hard links are counted (and displayed) a single time per *du* execution.

NOTES

Because symbolic links are not supported in this version of INTERIX, the **–H**, **–L**, and **–P** options have no effect.

Because filesystem mount points are not supported in this version of INTERIX, the **–x** behavior is the default.

SEE ALSO

df

NAME

echo – write arguments to standard output

SYNOPSIS

`echo [ -n ] args ...`

DESCRIPTION

Echo prints its arguments on the standard output, separated by spaces. Unless the **–n** option is present, a newline is output following the arguments. *echo* treats escape sequences specially, as described in the following paragraph.

If any of the following sequences of characters is encountered during output, the sequence is not output. Instead, the specified action is performed:

\b	Output a backspace character
\c	Subsequent output is suppressed. This is normally used at the end of the last argument to suppress the trailing newline that *echo* would otherwise output.
\f	Output a form feed.
\n	Output a newline character.
\r	Output a carriage return.
\t	Output a (horizontal) tab character.
\v	Output a vertical tab.
\0*ddd*	Output the character whose value is given by zero to three digits. If there are zero digits, a NUL character is output.
\\	Output a backslash.

Echo is implemented as a shell regular builtin.

EXAMPLES

Remember that backslash is special to the shell and needs to be escaped:

```
$ echo Here\'s a backslash: \\\\
```

To output a message to standard error, enter

```
$ echo message >&2
```

NAME

ed – line oriented text editor

SYNOPSIS

ed **[***-p prompt-string***] [***-s***] [***-v***] [***filename***]**

DESCRIPTION

Ed is a powerful line oriented editor. Although *ex* and *vi* have gained popularity, *ed* still maintains advantages over them. Most notable points are the **W** command (see below) (which is not part of *ex* or *vi*), the smaller executable size (you can often be editing before the others finish loading), and the better response when editing from slow terminals or across low baud data lines. *Ed* continues to be used by many system utilities.

When a *filename* is present, *ed* starts by simulating an **e** command (see below) If no filename is present *ed* starts with an empty buffer.

Options

Ed accepts the following options:

–p *prompt-string*
: Sets the prompt to *prompt-string*. The default prompt is *. (See also the **P** command.)

–s
: Suppress the printing of explanatory output from the commands **e**, **E**, **r**, **w**, **W** and **wq**. This option should be used with a script.

–v
: Display a message that describes which of BSD or POSIX mode *ed* has been set. This is useful for determining the described behavior below.

Behavior

Ed performs all changes to a copy of the file which is contained in a *buffer*. For the changes to have an effect one of the write commands (**w**, **W**, **wq**, or **Wq**) must be issued.

The contents of the *buffer* can changed by issuing commands that are lead by zero, one, or two addresses. All commands are alphabetically listed below with their parameter structures if applicable; trailing structures not described with commands are regarded as erroneous. Commands that accept zero addresses regard the presence of any address as an error.

Ed works in two modes: command, and input. The two modes are exclusive of each other. While in command mode *ed* accepts commands that display, modify, or give information about the *buffer*. While in input mode *ed* accepts lines of text to be added to the *buffer*.

Addressing in *ed* specifies one or more lines contained in the *buffer*. For commands that expect at least one address, and none are given, default addresses will be used. Using addresses in *ed* involves understanding that during the execution of most *ed* commands that a *current line* (*current*) exists. *Current* is usually the location in the *buffer* that the last command issued affected, although some commands do not affect *current*. Each command description (below) describes its affects on *current*.

Addresses

Addresses can be divided into three cases: one address (*single address*), two addresses (an *address pair*), and special address forms.

For the first two cases an address is formed with the use of:

1. A positive decimal integer (e.g. 123) indicating a line number in the buffer. Line number 1 is the first line in the buffer.
2. The '.' character indicating the current line (*current*).
3. The '$' character which indicates the last line in the buffer.
4. A regular expression (RE) enclosed with '/'s as delimiters (i.e. /RE/). This causes a forward search to the first occurrence of the specified RE. The address will then become this line. The character sequence ''\/'' escapes the forwardslash from being a delimiter. The search will wrap from the bottom of the buffer to the top of the buffer if need be. *Ed* RE's are, outside of this document, now referred to as *basic regular expressions*. Basic regular expressions (BRE's) are those that have traditionally worked in *ed*. BRE's are, for the most part, the same as the old RE's—the name has changed and the expressions extended to meet POSIX 1003.2 specifications. (See the search command for more details.)
5. A RE enclosed with '?'s as delimiters (i.e. ?RE?). This will cause a backward search to the first occurrence of the specified BRE. The address will then become this line. The character sequence \? escapes the questionmark from being a delimiter. The search will wrap from the top of the buffer to the bottom of the buffer if need be. (See the search command for more details.)
6. A line previously marked by the **k** command (see below). **'***x* addresses the line marked by the single lower-case letter '*x*' (from the portable character set in the range a-z).
7. An address of the form 1-6 followed by a '+' followed by an integer number, *n*, specifies the line to be addressed is *n* lines after the address of the form 1-6. If the address starts with a '+' then by default the addressed line is taken with respect to *current* (equivalent to '.' ; form 2). If no integer number is given then 1 is added to the address. Hence, if more than one '+' is given in a sequence, with no integer number following, 1 is added to the address for each '+'. Therefore, +++ is equivalent to +3, but +++1 is equivalent to +1.
8. An address of the form 1-6 followed by a '–' followed by an integer number, *n*, specifies the line to be addressed is *n* lines before the address of the form 1-6. If the address starts with a '–' then by default the addressed line is taken with respect to *current* ('.' ; form 1). If no integer number is given then 1 is subtracted from the address. Hence, if more than one '–' is given in a sequence, with no integer number following, 1 is subtracted from the address for each '–'. Therefore, —— is equivalent to –3, but ——1 is equivalent to –1. For backward compatibility '^' is the equivalent to '–'.
9. A ',' (comma) may be used to separate two addresses of the form 1-8 to create an *address pair*. The first address must occur no later in the buffer than the second address to be legal.
10. A ';' (semicolon) may be used to separate two addresses of the form 1-8 to create an *address pair*. With this form the second address is evaluated with respect to and

after the first address has been evaluated. This is useful when addresses of the forms 2-8 are used. The first address must occur no later in the buffer than the second address to be legal.

Note: Addresses of the forms 7 and 8 cannot be followed by addresses of forms 2-6; it is an error.

The following are special address forms that cannot be combined with any of the address forms listed above. A ',' by itself represents the address pair '1,$'. Likewise '%' by itself represents the address pair '1,$'. A ';' by itself represents the address pair '.,$'.

Ed Commands

The *ed* commands listed below default to the addresses prefixing the commands. Commands without default addresses accept zero addresses. The parentheses with the default addresses are not part of the address; they are used to show that the addresses are default.

Generally only one command appears on a line at a time. However, many of the commands may be suffixed by **l**, **n**, or **p**, in which case the current line is printed in the manner discussed below. These suffixes may be combined in any order.

(.) **a**
text
.
Append text after the addressed line. A '.' in the first column followed immediately by a newline places *ed* back in command mode — the '.' is not included in the text. Line 0 is legal for this command; text will be placed at the top of the buffer. *Current* is the last line appended (or the addressed line if no text given).

(.,.) **c**
text
.
Change text on the addressed line(s). The addressed lines are deleted before *ed* is placed in input mode. A '.' in the first column followed immediately by a newline places *ed* back in command mode—the '.' is not included in the text. *Current* is the new last line appended (or if no text is given the line after the addressed line deleted).

(.,.) **d** Delete the addressed line(s) from the buffer. Deleted lines may be recovered with the undo command (**u**; see below). *Current* is the line after the last addressed line deleted.

e [*filename*]
Edit the new file 'filename'. The *buffer* is cleared and the new file is placed in the *buffer*. If the *buffer* has been modified since the last write command *ed* will issue a warning ('?'); a second issuing of the command will be obeyed regardless. The number of characters read is printed (unless **–s** is specified at startup). If 'filename' is missing, the remembered name is used. If 'filename' is lead by ! then it shall be interpreted as a shell command to be executed, from which the standard output will be placed into the buffer; 'filename' will be non-remembered. Undo will not restore the *buffer* to its state before the edit command. *Current* is the last line in the *buffer* ('$').

E [*filename*]
E works the same as **e** except if the buffer has been modified no warning is issued.

f [*filename*]

Print the *remembered filename*. If *filename* is specified the *remembered filename* will be set to *filename*. If *filename* is lead by **!** then it is interpreted as a shell command to be executed, from which the standard output will be used as the new remembered filename. *Current* is unchanged.

(1,$) **g/***regular expression/command list*

The global command first marks all lines matching the regular expression. For each matching line, the command list is executed. At the start of each command list execution, *current* is set to equal that line; *current* may change as each command in the command list is executed for that line. The first command of the command list begins on the same line as the global command. Generally, in the command list one command occupies a line. Thus to have multiple commands in the command list it is necessary to escape the newline at the end of each line so that the global command does not interpret it as an indication that the command list entry has ended. The newline is escaped by proceeding it with a backslash (‘\’). Similarly with the commands that set *ed* into input mode the newlines of the entered text need to be escaped. If the ‘.’ used to end input mode is the last line of the command list the newline following the ‘.’ need not be escaped, or the ‘.’ may be omitted entirely. Commands in the command list can affect any line in the buffer. For the behaviour of each *ed* command within a command list refer to the information on the individual command, particularly **s** and **!**. The commands **g**, **G**, **v**, **V**, and **!** are permitted in the command list, but should be used with caution. The command list defaults to **p** if left empty (i.e. g / RE / p). For the regular expression the delimiters can be any characters except for space and newline; delimiters within a regular expression can be escaped with a backslash preceding it.

(1,$) **G/***regular expression/*

The interactive global command works similar to **g**. The first step is to mark every line which matches the given regular expression. For every line matched it will print this line, set *current* to this line, and accept one command (not including **a**, **c**, **i**, **g**, **G**, **v**, and **V**) for execution. The command can affect any line in the buffer. ‘%’ by itself executes the last non-null command. A return by itself will act as a null command. *Current* will be set to the last line affected by the last successful command input. If no match or an input command error occurs *current* will be set to the last line searched by **G**. **G** can be prematurely ended by <CTRL>-C (SIGINT). For the behaviour of each *ed* command within a command list refer to the information on the individual command, particularly **s** and **!**.

h The help command displays a message explaining the most recent command error (indicated by ‘?’). *Current* is unchanged.

H This toggles on or off the automatic display of messages explaining the most recent command error in place of ‘?’. *Current* is unchanged.

(.) **i**
text
.

The insert command places *ed* in input mode with the text being placed before the addressed line. Line 0 is invalid for this command. A ‘.’ in the first column followed immediately by a return places *ed* back in command mode—the ‘.’ is not included in the text. *Current* is the last line inserted. If no text is inserted

then *current* is the addressed line.

(.,.+1)**j** The join command joins the addressed lines together to make one line. If no addresses are specified, the *current* and *current*+1 lines are joined. If one address only is given then no join is performed. *Current* is unchanged.

(.)**k***x* The mark command marks the addressed line with label *x*, where *x* is a lowercase letter from the portable character set (a-z). The address form **'***x* will refer to this line (address form 6 above). *Current* is unchanged.

(.,.)**l** The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in three-digit octal preceded by a \ unless they are one of the following in which case they will be printed as indicated in the brackets: backslash (\\), horizontal tab (\t), form feed (\f). return (\r), vertical tab (\v), and backspace (\b). Long lines will be broken based on the type of terminal currently in use and will likely be ragged at the right side if text and octal are mixed on the same line. *Current* is set to the last line printed. The **l** command may be placed on the same line after any command except (**e**, **E**, **f**, **q**, **Q**, **r**, **w**, **W**, or !).

(.,.)**m***a* The move command moves the addressed lines in the buffer to after the address *a*. Line 0 is valid as the address **a** for this command. *Current* is the location in the *buffer* of the last line moved.

(.,.)**n** The number command prints the addressed lines preceding the text with the line number. The **n** command may be placed on the same line after any command except (**e**, **E**, **f**, **q**, **Q**, **r**, **w**, **W**, or !). *Current* is the last line printed.

(.,.)**p** The print command prints the addressed lines. The **p** command may be placed on the same line after any command except (**e**, **E**, **f**, **q**, **Q**, **r**, **w**, **W**, or !). *Current* is the last line printed.

(.,.)**P** The prompt is toggled on or off. *Current* is unchanged. The default prompt is "*" if not specified with the **–p** option at startup. The prompt is initially off unless the **–p** option is specified.

q The quit command causes *ed* to exit. If the entire *buffer* (1,$) has not been written since the last modification *ed* will issue a warning once ('?'); a second issuing of the command will be obeyed regardless.

Q **Q** works the same as **q** except if the buffer has been modified no warning is issued.

($)**r** [*filename*]
The read command reads in the file *filename* after the addressed line. If no *filename* is specified then the *remembered filename* is used. Address 0 is valid for this command. If read is successful then the number of characters read is printed (unless the **–s** option is specified). If 'filename' is lead by ! then it shall be interpreted as a shell command to be executed, from which the standard output will be placed in the buffer; 'filename' will be non-remembered. *Current* is the last line read.

(**.**,**.**)**s**/*regular expression*/*replacement*/*flags*
The substitute command searches for the regular expression in the addressed lines. On each line in which a match is found, matched strings are replaced by the *replacement* as specified by the *flags* (see below). If no *flags* appear, by default only the first occurrence of the matched string in each line is replaced.

It is an error if no matches to the RE occur.

The delimiters may be any character except space or newline. The delimiter lead by a \ will escape it to be a literal in the RE or *replacement*.

An ampersand, '&', appearing in the replacement will equal the string matching the RE. The '&'s special meaning can be suppressed by leading it with a '\'. When '%' is the only replacement character in *replacement*, the most recent replacement is used. The '%'s special meaning can be suppressed by leading it with a '\'.

The characters '*n*' (where *n* is a digit 1-9) is replaced by the text matching the RE subexpression *n* (known as backreferencing). **S** may be used to break lines by including a newline in *replacement* preceded by a backslash ('\') to escape it. *Replacement* can continue on the next line and can include another escaped newline.

The following extension should not be included in portable scripts. When splitting lines using **s** within the global commands (**g**, **G**, **v**, or **V**) the newline in the replacement string must be escaped by preceding it with '\ \ \' (three adjacent '\' s — the first '\' escapes the second '\' so that it is passed to **s** to escape the newline passed by the global command; the third '\' is to escape the newline so that the global command list continues). (N.B. Other *ed* implementations do not allow line splitting within the global commands.)

The *flags* may be any combination of:

count	in each addressed line replace the *count*–th matching occurrence.
g	in each addressed line replace all matching occurrences. When *count* and g are specified together inclusively replace in each addressed line all matches from the *count*-th match to the end of line.
l	write the line after replacement in the manner specified by the **l** command.
n	write the line after replacement in the manner specified by the **n** command.
p	write the line after replacement in the manner specified by the **p** command.

The following special form should not be included in portable scripts. This form is maintained for backward compatibility and is extended to dovetail into the above forms of **s**. **S** followed by *no* delimiters repeats the most recent substitute command on the addressed lines. **S** may be suffixed with the letters **r** (use the most recent RE rather than the last RE used with **s**), **p** (complement the setting of the any print command (**l**, **n**, **p**) suffix from the previous substitution), **g** (complement the setting of the **g** suffix) or **N** (negate the previous *count* flag). These modifying letters may be combined in any order (N.B. multiple use of the modifying letters may cause them to be interpreted as delimiters).

Current is set to where the last replacement occurred (POSIX).

(.,.)t a

The transcribe command copies the addressed lines in the *buffer* to after the address **a**. Address 0 is valid as the address **a** for this command. *Current* is the last line transcribed.

(.,.) **u** The undo command nullifies the most recent *buffer* modifying command. Buffer modifying commands undo works on are **a**, **c**, **d**, **g**, **G**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, and **V**. Marks set by the **k** command will also be restored. All commands (including nested **g**, **G**, **v**, and **V** commands within the **g** or **v**) that undo works on are treated as a single buffer modification. *Current* is set to the line it addressed before the last buffer modification.

(1, $) **v**/*regular expression/command list*
The global non-matching command performs as the **g** command does except that the command list is executed for every line that does not match the RE.

(1, $) **V**/*regular expression/*
The interactive global non-matching command is the same as the **G** except that one command will be accepted as input with *current* initially set to every line that does not match the RE.

(1, $) **w** [*filename*]
The write command writes the addressed lines to the file *filename*. If no 'filename' is specified then the *remembered filename* is used. If no addresses are specified the whole *buffer* is written. If the command is successful, the number of characters written is printed (unless the **–s** option is specified). If *filename* is lead by **!** then it shall be interpreted as a shell command to be executed which will accept on its standard input the section of the buffer specified for writing. *Current* is unchanged.

(1, $) **W** [*filename*]
W works as the **w** command does except the addressed contents of the *buffer* are appended to *filename* (or the *remembered filename* if *filename* is not specified). If *filename* is lead by **!** then **W** will act exactly as the **w** command. *Current* is unchanged.

(1, $) **wq** [*filename*]
wq works as the **w** command does with the addition that *ed* exits immediately after the write is complete. *Current* is unchanged.

(1,$) **Wq** [*filename*]
Wq works as the **W** command does with the addition that *ed* exits immediately after the appended write is complete. *Current* is unchanged.

(.+1) **z**

(.+1) **zn** Scroll through the *buffer*. Starting from the addressed line (or *current*+1) print the next 22 (by default or *n*) lines. The *n* is a sticky value; it becomes the default number of lines printed for successive scrolls. *Current* is the last line printed.

($) **=** Print the number of lines in the *buffer*. If an address is provided (in the forms 1-8 above) then the line number for that line will be printed. *Current* is unchanged.

!*shell command*
The command after the **!** is executed by *sh* and the results are printed. A '!' is printed in the first column when execution has completed (unless the **–s** option has been specified). A '!' immediately after **!** repeats the last shell command. An unescaped '%' represents the remembered filename. Commands to *sh* can have several lines by escaping the newline with a '\' immediately before it. The line continuation character for *sh*, '\', can be included on a line provided that it is escaped by a '\' immediately before so that *ed* passes it literally to *sh*: '\\'. It is implicit that for the command line that the *sh* line continuation character is

on that the newline will be escaped (e.g. '\ \ \ newline'). This behavior can be used within global command lists. However, an additional '\' must be added so that the **!** command continuator is passed to **!**—it must occur immediately before the global command's continuator. Therefore, the **!** command continuation sequence in a global command list will appear as '\ \ \' (explanation as with **s**). The line continuation character for *sh* needs no additional escaping (since it it not dependant on newline being adjacent)—hence, the sequence in a global command list with a line continuation will appear as '\ \ \ \ \ newline'. *Current* is unchanged.

/regular expression/
?*regular expression***?**
The search command searches forward, '/', (or backward, '?') through the *buffer* attempting to find a line that matches the RE. The search will wrap to the top (or bottom) of the *buffer* if necessary. Search returns the line number that the match occurs on - combined with the null command (see below) this causes the line to be printed. *Current* is the matching line.

(.+1, .+1) newline
The null command is equivalent to asking for the line *current*+1 to be printed according to the **p** command. This is a useful command to quickly print the next couple of lines. If more than a couple of lines are needed the **z** command (see above) is much better to use. *Current* is the last line printed.

OTHER

If an interrupt signal (SIGINT) is sent, *ed* prints '?' and returns to command mode.

BSD command pairs (pp, ll, etc.) are permitted. Additionally any single print command may follow any of the non-I/O commands (I/O commands: e, E, f, r, w, W, wq, and !). This will cause the current line to be printed in the specified manner after the command has completed.

Previous limitations on the number of characters per line and per command list have been lifted; there is now no maximum. File name and path length is restricted to the maximum length that the current file system supports. The **undo** command now restores marks to affected lines. The temporary buffer method will vary dependent on the method selected at compile. Two methods work with a temporary file (stdio and db), while the third uses memory. The limit on the number of lines depends on the amount of memory.

FILES

/tmp/_bsd44_ed* The temporary file.

ed.hup ed.hup: the buffer is written to this file in the current directory if possible and in the *HOME* directory is not (if the signal SIGHUP (hangup) is received).

DIAGNOSTICS

'?name' for a file that is either inaccessible, does not exist, or is a directory. '?' for all other errors unless the help messages have been toggled on (with the **H** command) in which case a descriptive message will be printed.

EOF is treated as a newline during input so that characters after the last newline are included into the *buffer*; the message "newline added at end of line" is printed.

Ed returns an exit status of 0 on successful completion. A value >0 is returned to indicate an *ed* error: 1 for a command line error, 2 for HUP signal received, 4 for an *ed* command error; these error values will be OR'ed together when appropriate.

SEE ALSO

sed

NAME

editres – a dynamic resource editor for X Toolkit applications

SYNOPSIS

`editres [-`*`toolkitoption`* `...]`

OPTIONS

Editres accepts all of the standard X Toolkit command line options (see *X*). The order of the command line options is not important.

DESCRIPTION

Editres is a tool that allows users and application developers to view the full widget hierarchy of any X Toolkit client that speaks the Editres protocol. In addition editres will help the user construct resource specifications, allow the user to apply the resource to the application and view the results dynamically. Once the user is happy with a resource specification editres will append the resource string to the user's X Resources file.

USING EDITRES

Editres provides a window consisting of the following four areas:

Menu Bar
A set of popup menus that allow you full access to editres's features.

Panner The panner allows a more intuitive way to scroll the application tree display.

Message Area
Displays information to the user about the action that editres expects of her.

Application Widget Tree
This area will be used to display the selected client's widget tree.

To begin an *editres* session select the **Get Widget Tree** menu item from the command menu. This will change the pointer cursor to cross hair. You should now select the application you wish look at by clicking on any of its windows. If this application understands the editres protocol then *editres* will display the client's widget tree in its tree window. If the application does not understand the *editres* protocol *editres* will inform you of this fact in the message area after a few seconds delay.

Once you have a widget tree you may now select any of the other menu options. The effect of each of these is described below.

COMMANDS

Send Widget Tree
Allows the user to click on any client that speaks the editres protocol and receive its widget tree.

Refresh Widget Tree
Editres only knows about the widgets that exist at the present time. Many applications create and destroy widgets "on-the-fly". Selecting this menu item will cause *editres* to ask the application to resend its widget tree, thus updating its information to the new state of the application.

Example
Xman only creates the widgets for its *topbox* when it starts up. None of the widgets for the manual page window are created until the user actually clicks on the *Manual Page* button. If you retrieved xman's widget tree before the the manual page is active, you may wish to refresh the widget tree after the

manual page has been displayed. This will allow you to also edit the manual page's resources.

Dump Widget Tree to a File

For documenting applications it is often useful to be able to dump the entire application widget tree to an ascii file. This file can then be included in the manual page. When this menu item is selected a popup dialog is activated. Type the name of the file in this dialog, and either select *okay*, or type a carriage-return. Editres will now dump the widget tree to this file. To cancel the file dialog just select the *cancel* button.

Show Resource Box

This command will popup a resource box for the current client. This resource box (described in detail below) will allow the user to see exactly which resources can be set for the widget that is currently selected in the widget tree display. Only one widget may be currently selected, if greater or fewer are selected *editres* will refuse to pop up the resource box, and put an error message in the **Message Area**.

Set Resource

This command will popup a simple dialog box for setting an arbitrary resource on all selected widgets. You must type in the resource name, as well as the value. You can use the Tab key to switch between the resource name field the resource value field.

Quit Exits editres.

TREE COMMANDS

The **Tree** menu contains several commands that allow operations to be performed on the widget tree.

Select Widget in Client

This menu item allows you to select any widget in the application, editres will then highlight the corrosponding element the widget tree display. Once this menu item is selected the pointer cursor will again turn to a crosshair, and you must click any pointer button in the widget you wish to have displayed. Since some widgets are fully obscured by their children, it is not possible to get to every widget this way, but this mechanism does give very useful feedback between the elements in the widget tree and those in the actual client.

Select All

Unselect All

Invert All

These functions allow the user to select, unselect, or invert all widgets in the widget tree.

Select Children

Select Parents

These functions select the immediate parent or children of each of the currently selected widgets.

Select Descendants

Select Ancestors

These functions select all parents or children of each of the currently selected widgets. This is a recursive search.

Show Widget Names

Show Class Names

Show Widget IDs

Show Widget Windows

When the tree widget is initially displayed the labels of each widget in the tree correspond to the widget names. These functions will cause the label of **all** widgets in the tree to be changed to show the class name, IDs, or window associated with each widget in the application. The widget IDs, and windows are shown as hex numbers.

In addition there are keyboard accelerators for each of the Tree operations. If the input focus is over an individual widget in the tree, then that operation will only effect that widget. If the input focus is in the Tree background it will have exactly the same effect as the corresponding menu item.

The translation entries shown may be applied to any widget in the application. If that widget is a child of the Tree widget, then it will only affect that widget, otherwise it will have the same effect as the commands in the tree menu.

Flash Active Widgets

This command is the inverse of the **Select Widget in Client** command, it will show the user each widget that is currently selected in the widget tree, by flashing the corresponding widget in the application *numFlashes* (three by default) times in the *flashColor*.

Key	Option	Translation Entry
space	**Unselect**	Select(nothing)
w	**Select**	Select(widget)
s	**Select**	Select(all)
i	**Invert**	Select(invert)
c	**Select Children**	Select(children)
d	**Select Descendants**	Select(descendants)
p	**Select Parent**	Select(parent)
a	**Select Ancestors**	Select(ancestors)
N	**Show Widget Names**	Relabel(name)
C	**Show Class Names**	Relabel(class)
I	**Show Widget IDs**	Relabel(id)
W	**Show Widget Windows**	Relabel(window)
T	**Toggle Widget/Class Name**	Relabel(toggle)

Clicking button 1 on a widget adds it to the set of selected widgets. Clicking button 2 on a widget deselects all other widgets and then selects just that widget. Clicking button 3 on a widget toggles its label between the widget's instance name the widget's class name.

USING THE RESOURCE BOX

The resource box contains five different areas. Each of the areas, as they appear on the screen, from top to bottom will be discussed.

The Resource Line This area at the top of the resource box shows the current resource name exactly as it would appear if you were to save it to a file or apply it.

The Widget Names and Classes

This area allows you to select exactly which widgets this resource will apply to. The area contains four lines, the first contains the name of the selected widget and all its ancestors, and the more restrictive dot (**.**) separator. The second line contains less specific the Class names of each widget, and well as the less restrictive star (*****) separator. The third line contains a set of special buttons called **Any Widget** which will generalize this level to match any widget. The last line contains a set of special buttons called **Any Widget Chain** which will turn the single level into something that matches zero or more levels.

The initial state of this area is the most restrictive, using the resource names and the dot separator. By selecting the other buttons in this are you can ease the restrictions to allow more and more widgets to match the specification. The extreme case is to select all the **Any Widget Chain** buttons, which will match every widget in the application. As you select different buttons the tree display will update to show you exactly which widgets will be effected by the current resource specification.

Normal and Constraint Resources

The next area allows you to select the name of the normal or constraint resources you wish to set. Some widgets may not have constraint resources, so that area will not appear.

Resource Value

This next area allows you to enter the resource value. This value should be entered exactly as you would type a line into your resource file. Thus it should contain no unescaped new-lines. There are a few special character sequences for this file:

`\n` This will be replaced with a newline.

`\###` Where # is any octal digit. This will be replaced with a single byte that contains this sequence interpreted as an octal number. For example, a value containing a NULL byte can be stored by specifying `\000`.

`\<new-line>` This will compress to nothing.

`\\` This will compress to a single backslash.

Command Area

This area contains several command buttons that I will describe in this section.

Set Save File

This button allows the user to modify file that the resources will be saved to. This button will bring up a dialog box that will ask you for a filename, once the filename has been entered, either hit carriage-return or click on the *okay* button. To popdown the dialog box without changing the save file, click the *cancel* button.

Save

This button will append the **resource line** described above to the end of the current save file. If no save file has been set the **Set Save File** dialog box will be popped up to prompt the user for a filename.

Apply

This button attempts to perform a XtSetValues call on all widgets that match

the **resource line** described above. The value specified is applied directly to all matching widgets. This behavior is an attempt to give a dynamic feel to the resource editor. Since this feature allows users to put an application in states it may not be willing to handle, a hook has been provided to allow specific clients block these SetValues requests (see **Blocking Editres Requests** below).

Unfortunately due to design constraints imposed on the widgets by the X Toolkit and the Resource Manager, trying to coerce an inherently static system into dynamic behavior can cause strange results. There is no guarantee that the results of an apply will be the same as what will happen when you save the value, and restart the application. This functionality is provided to try to give you a rough feel for what your changes will accomplish, and the results obtained should be considered suspect at best. Having said that, this is one of the neatest features of editres, and I strongly suggest that you play with it, and see what it can do.

Save and Apply
This button combines the Save and Apply actions described above into one button.

Popdown Resource Box
This button will remove the resource box from the display.

BLOCKING EDITRES REQUESTS

The *editres* protocol has been built into the Athena Widget set. This allows all application that are linked against Xaw to be able to speak to the resource editor. While this provides great flexability, and is a useful tool, it can quite easily be abused. It is therefore possible for any Xaw client to specify a value for the **editresBlock** resource described below, to keep *editres* from divulging information about its internals, or to disable the **SetValues** part of the protocol.

editresBlock (Class *EditresBlock)*
Specifies which type of blocking this client wishes to impose on the editres protocol.

The accepted values are:

all Block all requests.
setValues
Block all setvalues request, this is the only *editres* request that actually modifies the application, this is in effect stating that the applicaion is read-only.
none Allow all *editres* requests.

Remember that these resources are set on *any* Xaw client, not *editres*. They allow individual clients to keep all or some of the requests *editres* makes from ever succeeding. Of course, *editres* is also an Xaw client, so it may also be viewed and modified by *editres* (rather recursive, I know), these commands can be blocked by setting the **editresBlock** resource on *editres* itself.

RESOURCES

For *editres* the available application resources are:

numFlashes (Class **NumFlashes)**
Specifies the number of times the widgets in the client application will be flashed when the **Show Active Widgets** command in invoked.

flashTime (Class **FlashTime)**
Amount of time between the flashes described above.

flashColor (Class **flashColor)**
Specifies the color used to flash client widgets. A bright color should be used that will immediately draw your attention to the area being flashed, such as red or yellow.

saveResourcesFile (Class **SaveResourcesFile)**
This is the file the resource line will be append to when the **Save** button activated in the resource box.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *editres*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
Editres  editres
   Paned  paned
      Box  box
         MenuButton  commands
            SimpleMenu  menu
            SmeBSB  sendTree
            SmeBSB  refreshTree
            SmeBSB  dumpTreeToFile
            SmeLine  line
            SmeBSB  getResourceList
            SmeLine  line
            SmeBSB  quit
         MenuButton  treeCommands
            SimpleMenu  menu
            SmeBSB  showClientWidget
            SmeBSB  selectAll
            SmeBSB  unselectAll
            SmeBSB  invertAll
            SmeLine  line
            SmeBSB  selectChildren
            SmeBSB  selectParent
            SmeBSB  selectDescendants
            SmeBSB  selectAncestors
            SmeLine  line
            SmeBSB  showWidgetNames
            SmeBSB  showClassNames
            SmeBSB  showWidgetIDs
            SmeBSB  showWidgetWindows
            SmeLine  line
            SmeBSB  flashActiveWidgets
      Paned  hPane
         Panner  panner
         Label  userMessage
         Grip  grip
      Porthole  porthole
         Tree  tree
            Toggle  <name of widget in client>
```

```
                .
                TransientShell  resourceBox
                Paned  pane
                Label  resourceLabel
                Form  namesAndClasses
                Toggle  dot
                Toggle  star
                Toggle  any
                Toggle  name
                Toggle  class
                     .
                Label  namesLabel
                List  namesList
                Label  constraintLabel
                List  constraintList
                Form  valueForm
                Label  valueLabel
                Text  valueText
                Box  commandBox
                Command  setFile
                Command  save
                Command  apply
                Command  saveAndApply
                Command  cancel
                Grip  grip
        Grip  grip
```

ENVIRONMENT VARIABLES

DISPLAY
to get the default host and display number.

XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

FILES

/usr/lib/X11/app-defaults/Editres
specifies required resources.

SEE ALSO

X, *xrdb*

Athena Widget Set

RESTRICTIONS

This is a prototype, there are lots of nifty features I would love to add, but I hope this will give you some ideas about what a resource editor can do.

COPYRIGHT

Copyright 1990, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Chris D. Peterson, formerly MIT X Consortium

NAME

elvis – Visual editor emulating vi and ex

SYNOPSIS

```
elvis [-eirRsv] [+cmd] [-c cmd] [-m [file]] [-t tag]
      [-w winsize] [files...]
```

DESCRIPTION

Elvis is a text editor which emulates *vi* and *ex*.

Elvis can be installed under the names *ex* and *vi* (replacing the *nvi* versions of *ex* and *vi*). You may also install *elvis* under the names *view* and *input*. If *elvis* is installed on an NTFS partition, these extra names can be links to *elvis*; see the *ln* shell command.

When invoked as *vi*, the program behaves exactly as though it was invoked as "elvis". However, if you invoke *elvis* as *view*, then the readonly option is set as though you had given it the **–R** flag. If you invoke the program as *ex*, then *elvis* will start up in the colon command mode instead of the visual command mode, as though you had given it the **–e** flag. If you invoke *elvis* as *input* or *edit*, then *elvis* will start up in input mode, as though the **–i** flag was given.

Elvis takes the following options:

+*cmd* *Elvis* executes the command *cmd* after loading the first file. A typical example would be "elvis +237 foo", which would cause *elvis* to start editing foo and then move directly to line 237. *Cmd* must be a colon command.

–c *cmd* The System V version of the **+** option. *elvis* executes *cmd* after loading the first file. A typical example would be "elvis -c 237 foo", which would cause *elvis* to start editing foo and then move directly to line 237. *Cmd* must be a colon command.

–e *Elvis* will start up in colon command mode ("*ex*" mode).

–i *Elvis* will start up in input mode.

–m [*file*] *Elvis* will search through *file* for something that looks like an error message from a compiler. It will then begin editing the source file that caused the error, with the cursor sitting on the line where the error was detected. If you don't explicitly name a *file*, then "errlist" is assumed.

–r To the real *vi*, this flag means that a previous edit should be recovered. *Elvis*, though, has a separate program, called *elvrec*, for recovering files. When you invoke *elvis* with **–r**, *elvis* will tell you to run *elvrec*.

–R This sets the "readonly" option, so you won't accidentally overwrite a file.

–s This set the "safer" option, which disables many potentially harmful commands. It has not been rigorously proven to be absolutely secure, however.

–t *tag* This causes *elvis* to start editing at the given *tag*.

–v *Elvis* will start up in visual command mode.

–w *winsize*
Sets the "window" option's value to *winsize*.

FILES

Elvis makes use of the following files:

/tmp/elv*
During editing, *elvis* stores text in a temporary file.

tags This is the database used by the *:tags* command and the **-t** option. It is usually created by the *ctags* program.

$HOME/.exrc
A file called **.exrc** in your home directory is executed as a series of *ex* commands. A file by the same name may be executed in the current directory, too.

ENVIRONMENT

TERM This is the name of your terminal's entry in the termcap or terminfo database. The list of legal values varies from one system to another.

TERMCAP
Optional. If your system uses termcap, and the *TERMCAP* variable is unset, then *elvis* will read your terminal's definition from **$OPENNT_ROOT/usr/share/termcap**. (The *OPENNT_ROOT* directory is the location where you installed the INTERIX product.) If *TERMCAP* is set to the full pathname of a file (starting with a '/') then *elvis* will look in the named file instead of **$OPENNT_ROOT/usr/share/termcap**. If *TERMCAP* is set to a value which doesn't start with a '/', then its value is assumed to be the full termcap entry for your terminal.

TERMINFO
Optional. If your system uses terminfo, and the *TERMINFO* variable is unset, then *elvis* will read your terminal's definition from the database in the **$OPENNT_ROOT/usr/share/terminfo** database. If *TERMINFO* is set, then its value is used as the database name to use instead of **$OPENNT_ROOT/usr/share/terminfo**.

LINES, COLUMNS
Optional. These variables, if set, will override the screen size values given in the termcap/terminfo for your terminal. On windowing systems such as X, *elvis* has other ways of determining the screen size, so you should probably leave these variables unset.

EXINIT Optional. This variable can hold EX commands which will be executed instead of the .exrc file in your home directory.

SHELL Optional. The *SHELL* variable sets the default value for the "shell" option, which determines which shell program is used to perform wildcard expansion in file names, and also which is used to execute filters or external programs. The default value is "$OPENNT_ROOT/bin/sh".

HOME This variable should be set to the name of your home directory. *elvis* looks for its initialization file there; if *HOME* is unset then the initialization file will not be executed.

TAGPATH
Optional. This variable is used by the "ref" program, which is invoked by the shift-K, control-], and :tag commands. See "ref" for more information.

SEE ALSO

ctags, *ref*, *elvprsv*, *elvrec*

Elvis - A Clone of Vi/Ex, the complete *elvis* documentation.

BUGS

There is no LISP support. Certain other features are missing, too.

Auto-indent mode is not quite compatible with the real *vi*. Among other things, 0^D and ^^D don't do what you might expect.

Long lines are displayed differently. The real *vi* wraps long lines onto multiple rows of the screen, but *elvis* scrolls sideways.

AUTHOR

Steve Kirkendall
kirkenda@cs.pdx.edu

Many other people have worked to port *elvis* to various operating systems. To see who deserves credit, run the **:version** command from within *elvis*, or look in the system-specific section of the complete documentation.

NAME

elvrec - Recover the modified version of a file after an Elvis editor crash

SYNOPSIS

`elvrec [`*`preservedfile`* `[`*`newfile`*`]]`

DESCRIPTION

If you're editing a file when *elvis* dies, the system crashes, or power fails, the most recent version of your text will be preserved. The preserved text is stored in a special directory; it does *not* overwrite your text file automatically.

The *elvrec* program locates the preserved version of a given file, and writes it over the top of your text file — or to a new file, if you prefer. The recovered file will have nearly all of your changes.

To see a list of all recoverable files, run *elvrec* with no arguments.

(Note: if you haven't set up a directory for file preservation, then you'll have to manually run the *viprsv* program instead of *elvrec*.)

FILES

$OPENNT_ROOT/usr/preserve/p*
: The text that was preserved when *elvis* died.

$OPENNT_ROOT/usr/preserve/Index
: A text file which lists the names of all preserved files, and the names of the **$OPENNT_ROOT/usr/preserve/p*** files which contain their preserved text.

ENVIRONMENT VARIABLES

elvrec makes use of the following environment variable.

TMP Directory for temporary files

BUGS

elvrec is very picky about filenames. You must tell it to recover the file using exactly the same pathname as when you were editing it. The simplest way to do this is to go into the same directory that you were editing, and invoke *elvrec* with the same filename as *elvis*. If that doesn't work, then try running *elvrec* with no arguments, to see exactly which pathname it is using for the desired file.

If you're editing a nameless buffer when *elvis* dies, then *elvrec* will pretend that the file was named "foo".

AUTHOR

Steve Kirkendall
kirkenda@cs.pdx.edu

SEE ALSO

elvis

NAME

env – set environment for command invocation

SYNOPSIS

env **[**–i**]** **[***name*=*value***]**... **[***command***]**

DESCRIPTION

The *env* utility invokes the *command* after first modifying the environment as specified by the *name*=*value* operands. If *env* is invoked without arguments, it writes the current environment to standard output.

The options are as follows:

–i Invoke the command with exactly the environment specified.

DIAGNOSTICS

The *env* utility exits 0 on success, and >0 if an error occurs. It exits with 126 if the command could be found but not invoked, and 127 if *command* could not be found.

SEE ALSO

ksh, *sh*

NAME

expand – expand tabs to spaces

SYNOPSIS

```
expand [-t tablist] [file ...]
expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]
```

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

The *expand* program accepts these options:

–t *tablist* Specify tab stops according to *tablist*. The *tablist* is one or more positive decimal integers. If there is only one number, tabs are set *tablist* columns apart instead of 8 (the default).

If more than one number is specified, they are taken as the positions of the tabstops. The numbers *must* be in ascending order.

–*tabstop* If a single *tabstop* argument is given, then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given (separated by blank characters or commas), then the tabs are set at those specific columns.

Any tab characters beyond the last specified tab in a multiple tabstop list are replaced by a single space.

DIAGNOSTICS

Expand exits with status 0 on success and >0 if an error occurred, such as a problem opening a *file*.

SEE ALSO

unexpand

NAME

expr – simple expression evaluator

SYNOPSIS

```
expr [--] expression
expr [-x]
```

DESCRIPTION

Expr parses an *expression* provided as arguments and prints the result on the standard output.

Expr recognizes three types of operators: relational operators, arithmetic operators and string operators. The following left-associative binary operators are listed from lowest to highest precedence:

expr **|** *expr*
: Yields the first *expr* if it is neither null nor 0, otherwise it yields the second *expr*.

expr **&** *expr*
: Yields the first *expr* if neither *expr* is null nor 0, otherwise it yields 0.

expr relop expr
: where *relop* is one of **<**, **<=**, **=**, **!=**, **>=**, or **>**, yields 1 if the indicated comparison is true, otherwise 0 if false. The comparison is numeric if the two expressions are integers, otherwise it is lexicographic.

expr **+** *expr*
expr **–** *expr*
: Addition or subtraction of the arguments.

expr ***** *expr*
expr **/** *expr*
expr **%** *expr*
: Multiplication, division, or remainder of the arguments.

expr **:** *expr*
: The *matching* operator compares the string first argument with the regular expression second argument; regular expression matching is the same as the *ed* editor. The \(\) pattern symbols can be used to select a portion of the first argument. Otherwise the matching operator yields the number of characters matched (0 on failure).

(*expr* **)**
: Parentheses for grouping expressions.

The options are as follows:

–x Display a usage message

DIAGNOSTICS

Expr exits with 0 if the expression is neither null nor 0, exits with 1 if the expression is null or 0, and exits >1 if there is an error in the expression.

EXAMPLES

A simple arithmetic calculation; the answer is 8, because division is performed first.

```
$ expr 4 + 8 / 2
8
```

Check to see if the value of variable *input* is “hello”:

```
$ expr "$input" = "hello"
```

List the number of characters in the domain name, taken from the output of *whoami*:

```
$ exp ‘whoami‘ : \[^+\]*
```

Extract the domain name from the output of *whoami*; both characters in “\(” need to be escaped for the shell. An alternate form using single quotes instead of backslashes is also shown:

```
$ expr "$(whoami)" : \\\(.*\\\)+.*
$ expr "$(whoami)" : '\(.*\)+.*'
```

When matching strings, *expr* is best for breaking a string into two parts; it’s not very useful for extracting a particular word.

NOTES

Many of the operators are special to the shell and have to be quoted.

Operators can’t be used as (string) operands.

The POSIX version of *expr* is slightly different from the traditional version. If the first operand is a negative number (for example, –2), it’s treated as a command-line option. To ensure that it is treated as an operand, you must precede it with ––:

```
expr -- -2 + -3
```

SEE ALSO

sh, *test*

NAME

false – return false value

SYNOPSIS

```
false
```

DESCRIPTION

False is usually used in a Bourne shell script. It tests for the appropriate status "false" before running (or failing to run) a list of commands.

DIAGNOSTICS

The *false* utility always exits with a value other than zero.

SEE ALSO

sh, *true*

NAME

file – determine file type

SYNOPSIS

```
file [-c ] [-z ] [-f namefile]
     [-m magicfile] file ...
```

DESCRIPTION

File tests each argument in an attempt to classify it. There are three sets of tests, performed in this order: filesystem tests, magic number tests, and language tests. The *first* test that succeeds causes the file type to be printed.

The type printed will usually contain one of the words "text" (the file contains only ASCII characters and is probably safe to read on an ASCII terminal), "executable" (the file contains the result of compiling a program for either a Microsoft operating system or for some other vendor's operating system), or "data" meaning anything else (data is usually 'binary' or non-printable). Exceptions are well-known file formats (core files, tar archives) that are known to contain binary data. When modifying the file **magic** or the program itself, **preserve these keywords**. People depend on knowing that all the readable files in a directory have the word ''text'' printed. Don't do as Berkeley did – change ''shell commands text'' to ''shell script''.

The options are:

–c Check the format of the magic file; output displays the parsed form of the file. This is usually used in conjunction with **–m** to debug a new magic file before installing it.

–f *namefile*
Specifies that the names of the files to be examined are to be read (one per line) from *namefile* before the argument list. Either *namefile* or at least one filename argument must be present; to test the standard input, use ''-'' as a filename argument.

–m *file* Look in *file* for magic numbers, instead of **$OPENNT_ROOT/usr/share/magic**.

–z Try to look inside compressed files.

The filesystem tests are based on examining the return from a *stat*( ) system call. The program checks to see if the file is empty, or if it's some sort of special file. Any known file types appropriate to the system you are running on (sockets, symbolic links, or named pipes (FIFOs) on those systems that implement them) are intuited if they are defined in the system header file **<sys/stat.h>**.

The magic number tests are used to check for files with data in particular fixed formats. The canonical example of this is a binary executable (compiled program) **a.out** file, whose format is defined in **<a.out.h>** and possibly **<exec.h>** in the standard include directory. These files have a 'magic number' stored in a particular place near the beginning of the file that tells the operating system that the file is a binary executable, and which of several types thereof. The concept of 'magic number' has been applied by extension to data files. Any file with some invariant identifier at a small fixed offset into the file can usually be described in this way. The information in these files is read from the magic file **magic**.

If an argument appears to be an ASCII file, *file* attempts to guess its language. The language tests look for particular strings (cf **<names.h>**) that can appear anywhere in the first few blocks of a file. For example, the keyword **.br** indicates that the file is most likely a troff input file, just as the keyword **struct** indicates a C program. These tests are less reliable than the previous two groups, so they are performed last. The language test routines also test for some miscellany (such as *tar* archives) and determine whether an unknown file should be labelled as 'ascii text' or 'data'.

FILES

The *file* command uses the following files:

$OPENNT_ROOT/usr/share/magic
Default list of magic numbers.

STANDARDS CONFORMANCE

This program is believed to exceed the System V Interface Definition of *file*, as near as one can determine from the vague language contained therein. Its behaviour is mostly compatible with the System V program of the same name. This version knows more magic, however, so it will produce different (albeit more accurate) output in many cases.

The one significant difference between this version and System V is that this version treats any white space as a delimiter, so that spaces in pattern strings must be escaped. For example,

```
>10    string language impress\   (imPRESS data)
```

in an existing magic file would have to be changed to

```
>10    string language\ impress   (imPRESS data)
```

In addition, in this version, if a pattern string contains a backslash, it must be escaped. For example

```
0      string        \begindata    Andrew Toolkit document
```

in an existing magic file would have to be changed to

```
0      string        \\begindata   Andrew Toolkit document
```

SunOS releases 3.2 and later from Sun Microsystems include a *file* command derived from the System V one, but with some extensions. My version differs from Sun's only in minor ways. It includes the extension of the '&' operator, used as, for example,

```
>16    long&0x7fffffff     >0            not stripped
```

MAGIC DIRECTORY

The magic file entries have been collected from various sources, mainly USENET, and contributed by various authors. Christos Zoulas (address below) will collect additional or corrected magic file entries. A consolidation of magic file entries will be distributed periodically.

The order of entries in the magic file is significant. Depending on what system you are using, the order that they are put together may be incorrect. If your old *file* command uses a magic file, keep the old magic file around for comparison purposes (rename it to **magic.orig**).

HISTORY

There has been a *file* command since at least Research Version 6 (man page dated January, 1975). The System V version introduced one significant major change: the

external list of magic number types. This slowed the program down slightly but made it a lot more flexible.

This program, based on the System V version, was written by Ian Darwin without looking at anybody else's source code.

John Gilmore revised the code extensively, making it better than the first version. Geoff Collyer found several inadequacies and provided some magic file entries. The program has undergone continued evolution since.

AUTHOR

Written by Ian F. Darwin, UUCP address {utzoo | ihnp4}!darwin!ian, Internet address ian@sq.com, postal address: P.O. Box 603, Station F, Toronto, Ontario, CANADA M4Y 2L8.

Altered by Rob McMahon, cudcv@warwick.ac.uk, 1989, to extend the '&' operator from simple 'x&y != 0' to 'x&y op z'.

Altered by Guy Harris, guy@auspex.com, 1993, to:

- put the ''old-style'' '&' operator back the way it was, because 1) Rob McMahon's change broke the previous style of usage, 2) the SunOS ''new-style'' '&' operator, which this version of *file* supports, also handles 'x&y op z', and 3) Rob's change wasn't documented in any case;
- put in multiple levels of '>';
- put in ''beshort'', ''leshort'', etc. keywords to look at numbers in the file in a specific byte order, rather than in the native byte order of the process running *file*.

Changes by Ian Darwin and various authors including Christos Zoulas (christos@ee.cornell.edu), 1990-1992.

LEGAL NOTICE

Copyright (c) Ian F. Darwin, Toronto, Canada, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993.

This software is not subject to and may not be made subject to any license of the American Telephone and Telegraph Company, Sun Microsystems Inc., Digital Equipment Inc., Lotus Development Inc., the Regents of the University of California, The X Consortium or MIT, or The Free Software Foundation.

This software is not subject to any export provision of the United States Department of Commerce, and may be exported to any country or planet.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

1. The author is not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.

4. This notice may not be removed or altered.

A few support files (*getopt*, *strtok*) distributed with this package are by Henry Spencer and are subject to the same terms as above.

A few simple support files (*strtol*, *strchr*) distributed with this package are in the public domain; they are so marked.

The files **<tar.h>** and **is_tar.c** were written by John Gilmore from his public-domain *tar* program, and are not covered by the above restrictions.

BUGS

There must be a better way to automate the construction of the Magic file from all the glop in Magdir. What is it? Better yet, the magic file should be compiled into binary (say, *ndbm*(3) or, better yet, fixed-length ASCII strings for use in heterogenous network environments) for faster startup. Then the program would run as fast as the Version 7 program of the same name, with the flexibility of the System V version.

File uses several algorithms that favor speed over accuracy, thus it can be misled about the contents of ASCII files.

The support for ASCII files (primarily for programming languages) is simplistic, inefficient and requires recompilation to update.

There should be an ''else'' clause to follow a series of continuation lines.

The magic file and keywords should have regular expression support. Their use of ASCII TAB as a field delimiter is ugly and makes it hard to edit the files, but is entrenched.

It might be advisable to allow upper-case letters in keywords for e.g., troff commands vs man page macros. Regular expression support would make this easy.

The program doesn't grok FORTRAN. It should be able to figure FORTRAN by seeing some keywords which appear indented at the start of line. Regular expression support would make this easy.

The list of keywords in **ascmagic** probably belongs in the Magic file. This could be done by using some keyword like '*' for the offset value.

Another optimization would be to sort the magic file so that we can just run down all the tests for the first byte, first word, first long, etc, once we have fetched it. Complain about conflicts in the magic file entries. Make a rule that the magic entries sort based on file offset rather than position within the magic file?

The program should provide a way to give an estimate of "how good" a guess is. We end up removing guesses (e.g. ''From '' as first 5 chars of file) because they are not as good as other guesses (e.g. ''Newsgroups:'' versus "Return-Path:"). Still, if the others don't pan out, it should be possible to use the first guess.

This program is slower than some vendors' file commands.

This manual page, and particularly this section, is too long.

AVAILABILITY

You can obtain the original author's latest version by anonymous FTP on **tesla.ee.cornell.edu** in the directory **/pub/file-X.YY.tar.gz**.

SEE ALSO

od, *strings*

magic

NAME

find – walk a file hierarchy

SYNOPSIS

`find [-diXx] [-f file] [file ...] expression`

DESCRIPTION

Find recursively descends the directory tree for each *file* listed, evaluating an *expression* (composed of the "primaries" and "operands" listed below) in terms of each file in the tree.

The options are as follows:

–X Warns of filenames containing delimiting characters used by *xargs*. If a filename contains any of the delimiting characters used by *xargs*, *find* displays a diagnostic message on standard error and skips the file. Delimiting characters include single (" ' ") and double (" " ") quotes, backslash (" \ "), space, tab and newline characters.

–d Performs a *depth–first* traversal, i.e. directories are visited in post-order and all entries in a directory are acted on before the directory itself. By default, *find* visits directories in pre-order, i.e. before their contents. Note, the default is *not* a breadth-first traversal.

–f Specifies the file hierarchy for *find* to traverse. File hierarchies may also be specified as the operands immediately following the options.

–i Ignores case in filenames. This is useful when dealing with some networked files systems with Windows NT.

–x Prevents *find* from descending into directories that have a device number different than that of the file from which the descent began.

PRIMARIES

–atime *n*

True if the difference between the file last access time and the time *find* was started, rounded up to the next full 24–hour period, is *n* 24–hour periods.

–ctime *n*

True if the difference between the time of last change of file status information and the time *find* was started, rounded up to the next full 24–hour period, is *n* 24–hour periods.

–exec *utility argument ...* **;**

True if the program named *utility* returns a zero value as its exit status. Optional arguments may be passed to the utility. The expression must be terminated by a semicolon (";"). If the string "{}" appears anywhere in the utility name or the arguments it is replaced by the pathname of the current file. Because the semicolon has special meaning to the shell, it must be escaped or quoted. *Utility* will be executed from the directory from which *find* was executed.

–group *gname*
True if the file belongs to the group *gname*. If *gname* is numeric and there is no such group name, then *gname* is treated as a group id.

–inum *n*
True if the file has file serial number *n*.

–links *n*
True if the file has *n* links.

–ls This primary always evaluates to true. The following information for the current file is written to standard output: its file serial number, size in 512–byte blocks, file permissions, number of hard links, owner, group, size in bytes, last modification time, and pathname. If the file is a block or character special file, the major and minor numbers will be displayed instead of the size in bytes. If the file is a symbolic link, the pathname of the linked-to file will be displayed preceded by "–>". The format is identical to that produced by "ls –dgils".

–mtime *n*
True if the difference between the file last modification time and the time *find* was started, rounded up to the next full 24-hour period, is *n* 24-hour periods.

–ok *utility argument ...* **;**
The **–ok** primary is identical to the **–exec** primary with the exception that *find* requests user affirmation for the execution of the utility by printing a message to the terminal and reading a response. If the response is other than "y" the command is not executed and the value of the *ok* expression is false.

–name *pattern*
True if the last component of the pathname being examined matches *pattern*. Special shell pattern matching characters ("[", "]", "*", and "?") may be used as part of *pattern*. These characters may be matched explicitly by escaping them with a backslash ("\"). The *pattern* should be quoted or escaped to prevent the shell from evaluating it; see the *EXAMPLES* section.

–newer *file*
True if the current file has a more recent last modification time than *file*.

–nouser True if the file belongs to an unknown user.

–nogroup
True if the file belongs to an unknown group.

–path *pattern*
True if the pathname being examined matches *pattern*. Special shell pattern matching characters ("[", "]", "*", and "?") may be used as part of *pattern*. These characters may be matched explicitly by escaping them with a backslash ("\"). Slashes ("/") are treated as normal characters and do not have to be matched explicitly.

–perm *mode*
The *mode* may be either symbolic (see *chmod*) or an octal number. If the mode is symbolic, a starting value of zero is assumed and the mode sets or clears permissions without regard to the process' file mode creation mask. If the mode is octal, only bits 07777 (S_ISUID | S_ISGID | S_IRWXU | S_IRWXG | S_IRWXO) of the file's mode bits participate in the comparison. If the mode is preceded by a dash ("–"), this primary evaluates to true if at least all of the bits

in the mode are set in the file's mode bits. If the mode is not preceded by a dash, this primary evaluates to true if the bits in the mode exactly match the file's mode bits. Note, the first character of a symbolic mode may not be a dash ("–").

–print This primary always evaluates to true. It prints the pathname of the current file to standard output. The expression is appended to the user specified expression if neither **–exec**, **–ls** or **–ok** is specified.

–prune This primary always evaluates to true. It causes *find* to not descend into the current file. The **–prune** primary has no effect if the **–d** option was specified.

–size *n*[**c**]
True if the file's size, rounded up, in 512–byte blocks is *n*. If *n* is followed by a "c", then the primary is true if the file's size is *n* bytes.

–type *t* True if the file is of the specified type. Possible file types are as follows:

b	block special
c	character special
d	directory
f	regular file
p	FIFO

–user *uname*
True if the file belongs to the user *uname*. If *uname* is numeric and there is no such user name, then *uname* is treated as a user id.

All primaries which take a numeric argument allow the number to be preceded by a plus sign ("+") or a minus sign ("–"). A preceding plus sign means "more than n", a preceding minus sign means "less than n" and neither means "exactly n" .

OPERATORS

The primaries may be combined using the following operators. The operators are listed in order of decreasing precedence.

(*expression*)
This evaluates to true if the parenthesized expression evaluates to true. Because parentheses have special meaning to the shell, they will need to be escaped or quoted in some manner.

!*expression*
This is the unary NOT operator. It evaluates to true if the expression is false. Because the exclamation mark may have special meaning to your shell, you may need to escape or quote it in some manner.

expression **–and** *expression*
The **–and** operator is the logical AND operator. As it is implied by the juxtaposition of two expressions it does not have to be specified. The expression evaluates to true if both expressions are true. The second expression is not evaluated if the first expression is false. It can be shortened to **–a**.

expression **-or** *expression*
The **–or** operator is the logical OR operator. The expression evaluates to true if

either the first or the second expression is true. The second expression is not evaluated if the first expression is true. The operator can be shortened to **–o**.

All operands and primaries must be separate arguments to *find*. Primaries which themselves take arguments expect each argument to be a separate argument to *find*.

DIAGNOSTICS

Find returns 0 if all *path*s were successfully traversed, and >0 otherwise.

EXAMPLES

The following examples are shown as given to the shell:

Print out a list of all the files whose names do not end in ".c".

```
find / \! -name "*.c" -print
```

Print out a list of all the files whose names do not end in ".c" or ".h".

```
find / \! \( -name "*.c" -a -name "*.h" \) -print
```

Print out a list of all the files owned by user "wnj" that are newer than the file "ttt".

```
find / -newer ttt -user wnj -print
```

Print out a list of all the files which are not both newer than "ttt" and owned by "wnj".

```
find / \! \( -newer ttt -user wnj \) -print
```

Print out a list of all the files that are either owned by "wnj" or that are newer than "ttt".

```
find / \( -newer ttt -or -user wnj \) -print
```

Copy all files older than "ttt" to **/tmp**.

```
find / \! -newer ttt -exec cp {} /tmp ";"
```

SEE ALSO

xargs

NAME

finger – user information lookup program

SYNOPSIS

`finger [-dhlops] [user ...] [user@host ...]`

DESCRIPTION

The *finger* displays information about the system users.

Options are:

–d Print the *user*'s name as a fully qualified domain name — in the Windows NT *domain+name* format. When this option is used, the "Idle" field is not shown.

–h When used in conjunction with the **–s** option, display the name of the remote host instead of the office location and office phone.

–l Print long format, displaying all of the information described for the **–s** option as well as the user's home directory, home phone number, login shell, mail status, and the contents of the files **.plan** and **.project** from the user's home directory.

If idle time is at least a minute and less than a day, it is presented in the form "hh:mm". Idle times greater than a day are presented as "d day[s]hh:mm".

Phone numbers specified as eleven digits are printed as "+N-NNN-NNN-NNNN". Numbers specified as ten or seven digits are printed as the appropriate subset of that string. Numbers specified as five digits are printed as "xN-NNNN". Numbers specified as four digits are printed as "xNNNN".

If write permission is denied to the device, the phrase "(messages off)" is appended to the line containing the device name. One entry per user is displayed with the **–l** option; if a user is logged on multiple times, terminal information is repeated once per login.

Mail status is shown as "No Mail." if there is no mail at all, "Mail last read DDD MMM ## HH:MM YYYY (TZ)" if the person has looked at their mailbox since new mail arriving, or "New mail received ...", " Unread since ..." if they have new mail.

–o When used in conjunction with the **–s** option, the office location and office phone information is displayed instead of the name of the remote host.

–s *Finger* displays the user's login name, real name, terminal name and write status (as a "*" before the terminal name if write permission is denied), idle time, login time, and either office location and office phone number, or the remote host. If **–h** is given, the remote host is printed. If **–o** is given, the office location and phone number is printed instead (the default).

Idle time is in minutes if it is a single integer, hours and minutes if a ":" is present, or days if a "d" is present. Login time is displayed as the dayname if less than six days, else month, day, hours and minutes, unless more than six months ago, in which case the year is displayed rather than the hours and minutes. Idle time is not shown if the **–d** option is given.

Unknown devices as well as nonexistent idle and login times are displayed as single asterisks.

–p Prevents the **–l** option of *finger* from displaying the contents of the **.plan** and **.project** files.

If no options are specified, *finger* defaults to the **–l** style output if operands are provided, otherwise to the **–s** style. Some fields may be missing, in either format, if information is not available for them.

If no arguments are specified, *finger* will print an entry for each user currently logged into the system.

Finger may be used to look up users on a remote machine. The format is to specify a *user* as "user@host", or "@host", where the default output format for the former is the **–l** style, and the default output format for the latter is the **–s** style. The **–l** option is the only option that may be passed to a remote machine.

FILES

If present, *finger* makes use of the following files:

.plan *finger* displays the contents of this file in the home directory of the targeted *user*.

.project *finger* displays the contents of this file in the home directory of the targeted *user*.

SEE ALSO

who

NAME

fingerd – remote user information server

SYNOPSIS

`fingerd [-s] [-l] [-p` *`filename`*`]`

DESCRIPTION

Fingerd is a simple protocol based on RFC1196 that provides an interface to the *name* and *finger* programs at several network sites. The *fingerd* program is supposed to return a friendly, human-oriented status report on either the system at the moment or a particular person in depth. There is no required format and the protocol consists mostly of specifying a single "command line."

Fingerd listens for TCP requests at port 79. Once connected it reads a single command line terminated by a CRLF which is passed to *finger*. *Fingerd* closes its connections as soon as the output is finished.

If the line is null (i.e. just a <CRLF> is sent) then *finger* returns a "default" report that lists all people logged into the system at that moment.

If a user name is specified (e.g. eric <CRLF> then the response lists more extended information for only that particular user, whether logged in or not. Allowable "names" in the command line include only "user names" on INTERIX.

The following options may be passed to *fingerd* as server program arguments in **$OPENNT_ROOT/etc/inetd.conf**:

–l Enable logging. The name of the host originating the query is reported via *syslog*( ) at LOG_NOTICE priority.

–p Use an alternate program as the local information provider. The default local program executed by *fingerd* is *finger*. Specifying a customized local server gives a system manager more control over what information is provided to remote sites.

–s Enable secure mode. Queries without a user name are rejected and forwarding of queries to other remote hosts is denied.

NOTES

This daemon must be invoked from *inetd*; it cannot be started as a service directly.

Connecting directly to the server from a TIP or an equally narrow-minded TELNET–PROTOCOL user program can result in meaningless attempts at option negotiation being sent to the server, which will foul up the command line interpretation. *Fingerd* should be taught to filter out IAC's and perhaps even respond negatively (IAC WON'T) to all option commands received.

SEE ALSO

finger, *inetd*

NAME

flip – do newline conversions between POSIX text files and MS–DOS

SYNOPSIS

```
flip –h

flip –u|–m [–bpvtsz ] [file ...]
```

DESCRIPTION

Flip is a file interchange program that converts text file formats between POSIX and MS–DOS. It converts lines ending with carriage–return (CR) and linefeed (LF) to lines ending with just linefeed, or vice versa.

If no *file* is specified, or if the *file* is specified as -, *flip* reads the standard input. When *flip* reads from standard input, it writes to standard output. If you mix files and standard input in the list of operands, files are converted in place and standard input is converted then written to standard output.

Flip has the following features.

- If a file contains isolated CR characters for underlining or overprinting, *flip* does not change them.
- When asked to convert a file to the same format that it already has, *flip* causes no change to the file. Thus to convert all files to POSIX format you can type

  ```
  $ flip –u *.*   (under MS–DOS)
  $ flip –u * *   (under POSIX)
  ```

  and all files will end up right, regardless of whether they were in MS-DOS or in POSIX format to begin with. This also works in the opposite direction.
- *Flip* preserves file timestamps. You can override this.
- *Flip* is written in C and will compile and run under MS-DOS / Turbo C, 4.3BSD, and System V.
- *Flip* accepts wildcards and multiple filenames on the command line.
- If a user interrupt aborts *flip*, it does not leave behind any garbage files or cause corruption of the files being converted.
- *Flip* will normally refuse to convert binary files. You can override this.
- When converting from MS-DOS to POSIX format, *flip* removes any trailing control-Z (the last character in the file), but leaves embedded control-Z characters unchanged. This minimizes the possibility of accidentally converting a binary file that contains a control-Z near the beginning. You can override this and ask *flip* to recognize the first control-Z found as end-of-file.
- *Flip* can be asked to strip the high (parity) bit as it converts a file.

Flip is normally invoked as:

```
flip -umhvtb file ...
```

One of **–u**, **–m**, or **–h** is required. Switches may be given separately or combined together after a dash. For example, the three command lines given below are equivalent:

```
flip –uvt *.c
flip –u –v –t *.c
flip –u –vt *.c
```

The meanings of the switches are as follows.

–b Convert binary files too (else binary files are left unchanged)

–h Display a help message.

–m Convert to MS-DOS format (lone LF => CR LF, lone CR unchanged).

–p Print all output to stdout. Files named on the command line are unchanged.

–q Quiet mode. Suppress all messages.

–s Strip high bit.

–t Touch files (don't preserve timestamps).

–u Convert to POSIX format (CR LF => LF, lone CR or LF unchanged, trailing control-Z removed, embedded control-Z unchanged).

–v Verbose option, print filenames to standard error as they are processed.

–z Truncate file at first control-Z encountered.

On systems that allow a program to know its own name, *flip* may be renamed (or linked) to a file called *toix* for conversion to POSIX format, or to a file called *toms* for conversion to MS-DOS format. When invoked with the name *toix* or *toms*, *flip* will act as if it were invoked with the **–u** or **–m** option respectively.

COPYRIGHT

Both this documentation and *flip* are Copyright 1989 Rahul Dhesi, all rights reserved. Permission is granted to copy, use, and distribute for any commercial or noncommercial purpose in accordance with the requirements of version 1.0 of the *GNU General Public license*.

Note: This software has not been endorsed by the Free Software Foundation, the creator of the GNU license, and I am not affiliated with that organization.

This version has been extended for the INTERIX product.

AUTHOR

Rahul Dhesi

NAME

fmt – simple text formatter

SYNOPSIS

`fmt [`*`goal`* `[`*`maximum`*`]] [`*`file ...`*`]`

DESCRIPTION

Fmt is a simple text formatter which reads the concatenation of input files (or standard input if none are given) and produces on standard output a version of its input with lines as close to the *goal* length as possible without exceeding the *maximum*. The *goal* length defaults to 65 and the *maximum* to 75. The spacing at the beginning of the input lines is preserved in the output, as are blank lines and interword spacing.

Fmt is meant to format mail messages prior to sending, but may also be useful for other simple tasks. For instance, within visual mode of the *vi* editor, you can reformat a paragraph, evening the lines with the command:

```
!}fmt
```

SEE ALSO

fold, *mailx*

NAME

fold – fold long lines for finite width output device

SYNOPSIS

`fold [-w width] file ...`

DESCRIPTION

Fold is a filter which folds the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum of 80 characters.

The options are as follows:

–w *width* Specifies a line width to use instead of the default 80 characters. If the input text had tabs, *width* should be a multiple of 8.

DIAGNOSTICS

The *fold* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

fmt

NAME

fsinfo – font server information utility

SYNOPSIS

`fsinfo [-server servername]`

DESCRIPTION

Fsinfo is a utility for displaying information about an X font server. It is used to examine the capabilities of a server, the predefined values for various parameters used in communicating between clients and the server, and the font catalogues and alternate servers that are available.

EXAMPLE

The following shows a sample produced by *fsinfo*.

```
name of server:     hansen:7000
version number:     1
vendor string:      Font Server Prototype
vendor release number:     17
maximum request size:      16384 longwords (65536 bytes)
number of catalogues:      1
       all
Number of alternate servers: 2
    #0 hansen:7001
    #1 hansen:7002
number of extensions:      0
```

ENVIRONMENT

FONTSERVER

To get the default fontserver.

SEE ALSO

fs, *fslsfonts*

COPYRIGHT

Copyright 1991, Network Computing Devices, Inc
See *X* for a full statement of rights and permissions.

AUTHOR

Dave Lemke, Network Computing Devices, Inc

NAME

fslsfonts - server font list displayer for X font server

SYNOPSIS

`fslsfonts [-`*`options ...`*`] [-fn `*`pattern`*`]`

DESCRIPTION

Fslsfonts lists the fonts that match the given *pattern*. The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

OPTIONS

–server host:*port*
This option specifies the X font server to contact.

–l[**l**[**l**]]
(Each option is an ell, not a one.) This option indicates that medium, long, and very long listings, respectively, should be generated for each font.

–m This option indicates that long listings should also print the minimum and maximum bounds of each font.

–C This option indicates that listings should use multiple columns. This is the same as **–n 0**.

–1 This option indicates that listings should use a single column. This is the same as **–n 1**.

–w *width*
This option specifies the width in characters that should be used in figuring out how many columns to print. The default is 79.

–n *columns* This option specifies the number of columns to use in displaying the output. By default, it will attempt to fit as many columns of font names into the number of character specified by **–w** *width*.

–u
This option indicates that the output should be left unsorted.

SEE ALSO

fs, *showfont*, *xlsfonts*

ENVIRONMENT VARIABLES

FONTSERVER
Specifies the default host and port to use.

BUGS

Doing ‘‘fslsfonts -l’’ can tie up your server for a very long time. This is really a bug with single-threaded non-preemptable servers, not with this program.

COPYRIGHT

Copyright 1991, Network Computing Devices, Inc See *X* for a full statement of rights and permissions.

AUTHOR

Dave Lemke, Network Computing Devices, Inc

NAME

fstobdf – BDF font generator

SYNOPSIS

`fstobdf -fn` *`fontname`* **[***`-option ...`***]**

DESCRIPTION

The *fstobdf* program reads a font from a font server and prints a BDF file on the standard output that may be used to recreate the font. This is useful in testing servers, debugging font metrics, and reproducing lost BDF files.

OPTIONS

–server *servername*
This option specifies the server from which the font should be read.

–fn *fontname*
This option specifies the font for which a BDF file should be generated.

SEE ALSO

fs, *fslsfonts*

COPYRIGHT

Copyright 1990, Network Computing Devices

Copyright 1990, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHOR

Olaf Brandt, Network Computing Devices
Dave Lemke, Network Computing Devices

Jim Fulton, MIT X Consortium

NAME

ftp – ARPANET file transfer program

SYNOPSIS

`ftp [-dginv] [`*`host`*`]`

DESCRIPTION

Ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

Options may be specified at the command line, or to the command interpreter.

–d Enables debugging.

–g Disables file name globbing.

–i Turns off interactive prompting during multiple file transfers.

–n Restrains *ftp* from attempting "auto-login" upon initial connection. If auto-login is enabled, *ftp* will check the **.netrc** (see below) file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

–v Verbose option forces *ftp* to show all responses from the remote server, as well as report on data transfer statistics.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user the prompt

```
ftp>
```

is provided to the user. The following commands are recognized by *ftp*:

! [*command* [*args*]]
: Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

$ *macro-name* [*args*]
: Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

account [*passwd*]
: Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

append *local-file* **[***remote-file*]
: Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any **ntrans** or **nmap** setting. File transfer uses the current settings for **type**, **format**, **mode**, and **structure**.

ascii Set the file transfer type to network ASCII. This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary Set the file transfer type to support binary image transfer.

bye Terminate the FTP session with the remote server and exit *ftp*. An end of file will also terminate the session and exit.

case Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

cdup Change the remote machine working directory to the parent of the current remote machine working directory.

chmod *mode file-name*
Change the permission modes of the file *file-name* on the remote sytem to *mode*.

close Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

cr Toggle carriage return stripping during ascii type file retrieval. Records are denoted by a carriage return / linefeed sequence during ascii type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform with the POSIX single linefeed record delimiter. Records on non–POSIX remote systems may contain single linefeeds; when an ascii type transfer is made, these linefeeds may be distinguished from a record delimiter only when **cr** is off.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string `-->`

dir [*remote-directory* [*local-file*]]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving *dir* output. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is **–,** output comes to the terminal.

disconnect
A synonym for **close**.

form *format*
Set the file transfer form to *format*. The default format is “file”.

get *remote-file* **[***local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current *case*, **ntrans**, and **nmap** settings. The current settings for **type**, **form**, **mode**, and **structure** are used while transferring the file.

glob Toggle filename expansion for **mdelete**, **mget** and **mput**. If globbing is turned off with **glob**, the file name arguments are taken literally and not expanded. Globbing for **mput** is done as in *csh*. For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and ftp server, and can be previewed by doing

```
mls remote-files –
```

Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring a *tar* archive of the subtree (in binary mode).

hash Toggle hash-sign (“#”) printing for each data block transferred. The size of a data block is 1024 bytes.

hashn Like **hash**, turns on the printing of information for each data block transferred. Unlike **hash**, **hashn** updates a numeric display of the number of bytes, which is more readable for large transfers. Both **hash** and **hashn** interact: The **hashn** setting controls whether the output is # characters or numbers. Turning **hashn** on and off also turns **hash** on and off; turning **hash** off turns off **hashn**. Turning **hash** on never turns on **hashn**.

help [*command*]

Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

idle [*seconds*]

Set the inactivity timer on the remote server to *seconds* seconds. If *seconds* is omitted, the current inactivity timer is printed.

lcd [*directory*]

Change the working directory on the local machine. If no *directory* is specified, the user’s home directory is used.

ls [*remote-directory* [*local-file*]]

Print a listing of the contents of a directory on the remote machine. The listing includes any system-dependent information that the server chooses to include; for example, most systems with an *ls* command will use output from the command *ls –l*. (See also **nlist**.) If *remote-directory* is left unspecified, the current working directory is used. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **ls** output. If no local file is specified, or if *local-file* is **–**, the output is sent to the terminal.

macdef*macro-name*

Define a macro. Subsequent lines are stored as the macro *macro-name;* a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets ‘$’ and ‘\’ as special characters. A ‘$’ followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A ‘$’ followed by an ‘i’ signals that macro processor that the executing macro is to be looped. On the first pass ‘$i’ is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A ‘\’ followed by any character is replaced by that character. Use the ‘\’ to prevent special treatment of the ‘$’.

mdelete [*remote-files*]

Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*

Like **dir**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*

Expand the *remote-files* on the remote machine and do a **get** for each file name

thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to **case**, **ntrans**, and **nmap** settings. Files are transferred into the local working directory, which can be changed with *lcd directory*; new local directories can be created with *! mkdir directory*.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like **nlist**, except multiple remote files may be specified, and the **local-file** must be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

mode [*mode-name*]
Set the file transfer mode to *mode-name*. The default mode is "stream" mode.

modtime *file-name*
Show the last modification time of the file on the remote machine.

mput *local-files*
Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to **ntrans** and **nmap** settings.

newer *file-name*
Get the file only if the modification time of the remote file is more recent that the file on the current system. If the file does not exist on the current system, the remote file is considered *newer*. Otherwise, this command is identical to **get.**

nlist [*remote-directory*] [*local-file*]
Print a list of the files in a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If the system has an *ls* command, that output will probably be used for the **nlist** command. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **nlist** output. If no local file is specified, or if *local-file* is **–**, the output is sent to the terminal.

nmap [*inpattern outpattern*]
Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences '$1', '$2', ..., '$9' in *inpattern*. Use '\' to prevent this special treatment of the '$' character. All other characters are treated literally, and are used to determine the **nmap** [*inpattern*] variable values. For example, given *inpattern* $1.$2 and the remote file name "mydata.data", $1 would have the value "mydata", and $2 would have the value "data". The *outpattern* determines the resulting mapped filename. The sequences '$1', '$2', ..., '$9' are replaced by any value resulting from the *inpattern* template. The sequence '$0' is replace by the original filename. Additionally, the sequence

[*seq1, seq2*] is replaced by [*seq1*] if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command

```
nmap $1.$2.$3 [$1,$2].[$2,file]
```

would yield the output filename "myfile.data" for input filenames "myfile.data" and "myfile.data.old", "myfile.file" for the input filename "myfile", and "myfile.myfile" for the input filename ".myfile". Spaces may be included in *outpattern*, as in the example: ‘nmap $1 sed "s/ *$/ /" > $1’ . Use the ‘\’ character to prevent special treatment of the ‘$’,’[’,’[’, and ‘,’ characters.

ntrans [*inchars* [*outchars*]]
Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a computer with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character’s position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

open *host* [*port*]
Establish a connection to the specified *host*’s FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the **auto-login** option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

proxy *ftp-command*
Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put, mput,** and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

put *local-file* **[***remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any **ntrans** or **nmap** settings in naming the remote file. File transfer uses the current settings for **type**, **format**, **mode**, and **structure**.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server.

recv *remote-file* [*local-file*]
A synonym for **get**.

reget *remote-file* **[***local-file*]
Reget acts like **get**, except that if *local-file* exists and is smaller than *remote-file*, *local-file* is presumed to be a partially transferred copy of *remote-file* and the transfer is continued from the apparent point of failure. This command is useful when transferring very large files over networks that are prone to dropping connections.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

remotestatus [*file-name*]
With no arguments, show status of remote machine. If *file-name* is specified, show status of *file-name* on remote machine.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

reset Clear reply queue. This command re-synchronizes command / reply sequencing with the remote ftp server. Resynchronization may be necessary following a violation of the ftp protocol by the remote server.

restart *marker*
Restart the immediately following **get** or **put** at the indicated *marker*. The *marker* is usually a byte offset into the file.

rmdir *directory-name*
Delete a directory on the remote machine.

runique
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a ".1" is appended to the name. If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that **runique** will not affect local files generated from a shell command (see below). The default value is off.

send *local-file* [*remote-file*]
A synonym for **put**.

sendport
Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

site *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server as a SITE command.

size *file-name*
Return size of *file-name* on remote machine.

status Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer structure to *struct-name.* By default "stream" structure is used.

sunique
Toggle storing of files on remote machine under unique file names. Remote ftp server must support ftp protocol STOU command for successful completion. The remote server will report unique name. Default value is off.

system Show the type of operating system running on the remote machine.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer type to **type-name.** If no type is specified, the current type is printed. The default type is network ASCII.

umask [*newmask*]
Set the default umask on the remote server to *newmask.* If *newmask* is omitted, the current umask is printed.

user *user-name* **[***password*] [*account*]
Identify yourself to the remote FTP server. If the *password* is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an *account* field is not specified, and the FTP server requires it, the user will be prompted for it. If an *account* field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless *ftp* is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose
Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
A synonym for **help**.

Command arguments which have embedded spaces may be quoted with quote '"' marks.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually Ctrl-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a ftp protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed by hand.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules.

1. If the file name '-' is specified, the stdin (for reading) or stdout (for writing) is used.
2. If the first character of the file name is '|', the remainder of the argument is interpreted as a shell command. *Ftp* then forks a shell, using *popen*( ) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. "\"ls -lt\"". A particularly useful example of this mechanism is:

   ```
   dir . |more
   ```

3. Failing the above checks, if ''globbing'' is enabled, local file names are expanded according to the rules used in the *csh*; c.f. the *glob* command. If the *ftp* command expects a single local file (.e.g. **put),** only the first filename generated by the "globbing" operation is used.
4. For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.
5. For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The **type** may be one of "ascii", "image" (binary), "ebcdic", and "local byte size" (for PDP-10's and PDP-20's mostly). *Ftp* supports the ascii and image types of file transfer, plus local byte size 8 for tenex mode transfers.

Ftp supports only the default values for the remaining file transfer parameters: **mode**, **form**, and **struct**.

THE .netrc FILE

The **.netrc** file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

machine *name*
: Identify a remote machine *name.* The auto-login process searches the **.netrc** file for a **machine** token that matches the remote machine specified on the *ftp* command line or as an **open** command argument. Once a match is made, the subsequent **.netrc** tokens are processed, stopping when the end of file is reached or another **machine** or a **default** token is encountered.

default
: This is the same as **machine** *name* except that **default** matches any name. There can be only one **default** token, and it must be after all **machine** tokens. This is normally used as:

  ```
  default login anonymous password user@site
  ```

  thereby giving the user *automatic* anonymous ftp login to machines not specified in **.netrc**. This can be overridden by using the **–n** flag to disable auto-login.

login *name*
: Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified *name.*

password *string*

Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the **.netrc** file for any user other than *anonymous, ftp* will abort the auto-login process if the **.netrc** is readable by anyone besides the user.

account *string*

Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

macdef *name*

Define a macro. This token functions like the *ftp* **macdef** command functions. A macro is defined with the specified name; its contents begin with the next **.netrc** line and continue until a null line (consecutive new-line characters) is encountered. If a macro named *init* is defined, it is automatically executed as the last step in the auto-login process.

ENVIRONMENT VARIABLES

Ftp utilizes the following environment variables.

HOME For default location of a **.netrc** file, if one exists.
SHELL For default shell.

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the BSD 4.2 ascii-mode transfer code. has been corrected. This may result in incorrect transfers of binary files to and from BSD 4.2 servers using the ascii type. Avoid this problem by using the binary image type.

NAME

ftpd – Internet File Transfer Protocol server

SYNOPSIS

`ftpd [-dl] [-T maxtimeout] [-t timeout]`

DESCRIPTION

Ftpd is the Internet File Transfer Protocol server process. The server uses the TCP protocol and listens at the port specified in the "ftp" service specification.

Available options:

–d Debugging information is written to the syslog using LOG_FTP.

–l Each successful and failed *ftp* session is logged using syslog with a facility of LOG_FTP. If this option is specified twice, the retrieve (get), store (put), append, delete, make directory, remove directory and rename operations and their filename arguments are also logged.

–T A client may also request a different timeout period; the maximum period allowed may be set to *timeout* seconds with the **–T** option. The default limit is 2 hours.

–t The inactivity timeout period is set to *timeout* seconds (the default is 15 minutes).

The file **/etc/nologin** can be used to disable ftp access. If the file exists, displays it and exits. If the file **/etc/ftpwelcome** exists, prints it before issuing the "ready" message. If the file **/etc/motd** exists, prints it after a successful login.

The ftp server currently supports the following ftp requests. The case of the requests is ignored.

Request	Description
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory (""ls -lgA"")
MKD	make a directory
MDTM	show last modification time of file
MODE	specify data transfer Em mode
NLST	give name list of files in directory
NOOP	do nothing
PASS	specify password
PASV	prepare for server-to-server transfer

PORT	specify data connection port
PWD	print the current working directory
QUIT	terminate session
REST	restart incomplete transfer
RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name
SITE	non-standard commands (see next section)
SIZE	return size of file
STAT	return status of server
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer Em structure
SYST	show operating system type of server system
TYPE	specify data transfer Em type
USER	specify user name
XCUP	change to parent of current working directory (deprecated)
XCWD	change working directory (deprecated)
XMKD	make a directory (deprecated)
XPWD	print the current working directory (deprecated)
XRMD	remove a directory (deprecated)

The following non-standard commands are supported by the SITE request.

Request	**Description**
UMASK	change umask, e.g. ‘‘SITE UMASK 002’’
IDLE	set idle-timer, e.g. ‘‘SITE IDLE 60’’
CHMOD	change mode of a file, e.g. ‘‘SITE CHMOD 755 filename’’
HELP	give help information.

The remaining ftp requests specified in Internet RFC 959 are recognized, but not implemented. MDTM and SIZE are not specified in RFC 959, but will appear in the next updated FTP RFC.

The ftp server will abort an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959. If a STAT command is received during a data transfer, preceded by a Telnet IP and Synch, transfer status will be returned.

Ftpd interprets file names according to the “globbing” conventions used by *csh*. This allows users to utilize the metacharacters “*?[]{}~.”

Ftpd authenticates users according to these rules:

1. The login name must be in the password data base, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.
2. The login name must not appear in the file **$OPENNT_ROOT/etc/ftpusers**.

Anonymous FTP

On many systems (but not INTERIX), there is a provision for "anonymous ftp" — if the login names "anonymous" or "ftp" match an account on the system, the user is allowed to login with any password. On these systems, the *ftpd* performs a *chroot*( ) system call and takes other measures to preserve security.

On INTERIX, there is **no** anonymous ftp; the user names "anonymous" or "ftp" have no special meaning. Because INTERIX provides no *chroot*( ) system call, anonymous ftp sessions are insecure and dangerous.

FILES

The *ftpd* makes use of the following files:

$OPENNT_ROOT/etc/ftpusers
: List of unwelcome / restricted users.

$OPENNT_ROOT/etc/ftpwelcome
: Welcome notice.

$OPENNT_ROOT/etc/motd
: Welcome notice after login.

$OPENNT_ROOT/etc/nologin
: Displayed and access refused.

BUGS

The possible security holes have been extensively scrutinized, but are possibly incomplete.

SEE ALSO

ftp, *inetd*, *syslogd*

NAME

g++ – GNU project C++ compiler

SYNOPSIS

g++ **[***option*|*filename***]** ...

DESCRIPTION

The C and C++ compilers are integrated; *g++* is a script to call *gcc with options to recognize C++*. *Gcc* processes input files through one or more of four stages: preprocessing, compilation, assembly, and linking. This man page contains full descriptions for only C++ specific aspects of the compiler, though it also contains summaries of some general-purpose options. For a fuller explanation of the compiler, see *gcc*.

C++ source files use one of the suffixes '.C', '.cc', or '.cxx'; preprocessed C++ files use the suffix '.ii'.

OPTIONS

There are many command-line options, including options to control details of optimization, warnings, and code generation, which are common to both *gcc* and *g++*. For full information on all options, see *gcc*.

Options must be separate: **–dr** is quite different from **–d –r**.

Most **–f** and **–W** options have two contrary forms: **–f***name* and **–f***no-name* (or **–W***name* and **–W***no-name*). Only the non-default forms are shown here.

–c Compile or assemble the source files, but do not link. The compiler output is an object file corresponding to each source file.

–D *macro*

Define macro *macro* with the string **–1** as its definition.

–D *macro***=***defn*

Define macro *macro* as *defn*.

–E Stop after the preprocessing stage; do not run the compiler proper. The output is preprocessed source code, which is sent to the standard output.

–fall-virtual

Treat all possible member functions as virtual, implicitly. All member functions (except for constructor functions and new or delete member operators) are treated as virtual functions of the class where they appear.

This does not mean that all calls to these member functions will be made through the internal table of virtual functions. Under some circumstances, the compiler can determine that a call to a given virtual function can be made directly; in these cases the calls are direct in any case.

–fdollars-in-identifiers

Permit the use of **–$** in identifiers. Traditional C allowed the character $ to form part of identifiers; by default, GNU C also allows this. However, ANSI C forbids $ in identifiers, and GNU C++ also forbids it by default on most platforms (though on some platforms it's enabled by default for GNU C++ as well).

–felide-constructors

Use this option to instruct the compiler to be smarter about when it can elide constructors. Without this flag, GNU C++ and cfront both generate effectively the same code for:

```
A foo ();
A x (foo ()); // x initialized by 'foo ()', no ctor called
A y = foo (); // call to 'foo ()' heads to temporary,
// y is initialized from the temporary.
```

Note the difference! With this flag, GNU C++ initializes **–y** directly from the call to foo () without going through a temporary.

–fenum-int-equiv

Normally GNU C++ allows conversion of enum to int, but not the other way around. Use this option if you want GNU C++ to allow conversion of int to enum as well.

–fexternal-templates

Produce smaller code for template declarations, by generating only a single copy of each template function where it is defined. To use this option successfully, you must also mark all files that use templates with either **#pragma implementation** (the definition) or **#pragma interface** (declarations).

When your code is compiled with **–fexternal-templates**, all template instantiations are external. You must arrange for all necessary instantiations to appear in the implementation file; you can do this with a typedef that references each instantiation needed. Conversely, when you compile using the default option **–fno-external-templates**, all template instantiations are explicitly internal.

–fno-gnu-linker

Do not output global initializations (such as C++ constructors and destructors) in the form used by the GNU linker (on systems where the GNU linker is the standard method of handling them). Use this option when you want to use a non-GNU linker, which also requires using the collect2 program to make sure the system linker includes constructors and destructors. (collect2 is included in the GNU CC distribution.) For systems which must use collect2, the compiler driver *gcc* is configured to do this automatically.

–fmemoize-lookups
–fsave-memoized

These flags are used to get the compiler to compile programs faster using heuristics. They are not on by default since they are only effective about half the time. The other half of the time programs compile more slowly (and take more memory).

The first time the compiler must build a call to a member function (or reference to a data member), it must (1) determine whether the class implements member functions of that name; (2) resolve which member function to call (which involves figuring out what sorts of type conversions need to be made); and (3) check the visibility of the member function to the caller. All of this adds up to slower compilation. Normally, the second time a call is made to that member function (or reference to that data member), it must go through the same lengthy process again. This means that code like this

```
cout << "This " << p << " has " << n << " legs.0;
```

makes six passes through all three steps. By using a software cache, a ‘‘hit’’ significantly reduces this cost. Unfortunately, using the cache introduces another layer of mechanisms which must be implemented, and so incurs its own overhead. **–fmemoize-lookups** enables the software cache.

Because access privileges (visibility) to members and member functions may differ from one function context to the next, *g++* may need to flush the cache. With the **–fmemoize-lookups** flag, the cache is flushed after every function that is compiled. The **–fsave-memoized** flag enables the same software cache, but when the compiler determines that the context of the last function compiled would yield the same access privileges of the next function to compile, it preserves the cache. This is most helpful when defining many member functions for the same class: with the exception of member functions which are friends of other classes, each member function has exactly the same access privileges as every other, and the cache need not be flushed.

–fno-default-inline

Do not make member functions inline by default merely because they are defined inside the class scope. Otherwise, when you specify **–O**, member functions defined inside class scope are compiled inline by default; i.e., you don’t need to add ‘inline’ in front of the member function name.

–fno-strict-prototype

Consider the declaration int foo ();. In C++, this means that the function foo takes no arguments. In ANSI C, this is declared int foo(void);. With the flag **–fno-strict-prototype**, declaring functions with no arguments is equivalent to declaring its argument list to be untyped, i.e., int foo (); is equivalent to saying int foo (...);.

–fnonnull-objects

Normally, GNU C++ makes conservative assumptions about objects reached through references. For example, the compiler must check that **–a** is not null in code like the following:

```
obj &a = g ();
a.f (2);
```

Checking that references of this sort have non-null values requires extra code, however, and it is unnecessary for many programs. You can use **–fnonnull-objects** to omit the checks for null, if your program doesn’t require the default checking.

–fhandle-signatures
–fno-handle-signatures

These options control the recognition of the signature and sigof constructs for specifying abstract types. By default, these constructs are not recognized.

–fthis-is-variable

The incorporation of user-defined free store management into C++ has made assignment to this an anachronism. Therefore, by default GNU C++ treats the type of this in a member function of class X to be X *const. In other words, it is illegal to assign to this within a class member function. However, for backwards compatibility, you can invoke the old behavior by using **–fthis-is-variable**.

–g Produce debugging information in the operating system's native format (for DBX or SDB or DWARF). GDB also can work with this debugging information. On most systems that use DBX format, **–g** enables use of extra debugging information that only GDB can use.

Unlike most other C compilers, GNU CC allows you to **–g** with **–O**. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

–I*dir* Append directory *dir* to the list of directories searched for include files.

–L*dir* Add directory *dir* to the list of directories to be searched for **–l**.

–l*library*

Use the library named *library* when linking. (C++ programs often require **–lg++** for successful linking.)

–nostdinc

Do not search the standard system directories for header files. Only the directories you have specified with **–I** options (and the current directory, if appropriate) are searched.

–nostdinc++

Do not search for header files in the standard directories specific to C++, but do still search the other standard directories. (This option is used when building libg++.)

–O Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

–o *file*

Place output in file file.

–S Stop after the stage of compilation proper; do not assemble. The output is an assembler code file for each non-assembler input file specified.

–traditional

Attempt to support some aspects of traditional C compilers.

Specifically, for both C and C++ programs:

- In the preprocessor, comments convert to nothing at all, rather than to a space. This allows traditional token concatenation.
- In the preprocessor, macro arguments are recognized within string constants in a macro definition (and their values are stringified, though without additional quote marks, when they appear in such a context). The preprocessor always considers a string constant to end at a newline.
- The preprocessor does not predefine the macro __STDC__ when you use **–traditional**, but still predefines__GNUC__ (since the GNU extensions indicated by __GNUC__ are not affected by **–traditional**). If you need to

write header files that work differently depending on whether **–traditional** is in use, by testing both of these predefined macros you can distinguish four situations: GNU C, traditional GNU C, other ANSI C compilers, and other old C compilers.

- In the preprocessor, comments convert to nothing at all, rather than to a space. This allows traditional token concatenation.
- In the preprocessor, macro arguments are recognized within string constants in a macro definition (and their values are stringified, though without additional quote marks, when they appear in such a context). The preprocessor always considers a string constant to end at a newline.
- The preprocessor does not predefine the macro __STDC__ when you use **–traditional**, but still predefines__GNUC__ (since the GNU extensions indicated by __GNUC__ are not affected by **–traditional**). If you need to write header files that work differently depending on whether **–traditional** is in use, by testing both of these predefined macros you can distinguish four situations: GNU C, traditional GNU C, other ANSI C compilers, and other old C compilers.
- String "constants" are not necessarily constant; they are stored in writable space, and identical looking constants are allocated separately.

For C++ programs only (not C), **–traditional** has one additional effect: assignment to this is permitted. This is the same as the effect of **–fthis-is-variable**.

–U*macro* Undefine macro macro.

–Wall Issue warnings for conditions which pertain to usage that we recommend avoiding and that we believe is easy to avoid, even in conjunction with macros.

–Wenum-clash
Warn when converting between different enumeration types.

–Woverloaded-virtual
In a derived class, the definitions of virtual functions must match the type signature of a virtual function declared in the base class. Use this option to request warnings when a derived class declares a function that may be an erroneous attempt to define a virtual function: that is, warn when a function with the same name as a virtual function in the base class, but with a type signature that doesn't match any virtual functions from the base class.

–Wtemplate-debugging
When using templates in a C++ program, warn if debugging is not yet fully available.

–w Inhibit all warning messages.

+eN Control how virtual function definitions are used, in a fashion compatible with *cfront* 1.*x*.

PRAGMAS

Two **#pragma** directives are supported for GNU C++, to permit using the same header file for two purposes: as a definition of interfaces to a given object class, and as the full definition of the contents of that object class.

#pragma interface

Use this directive in header files that define object classes, to save space in most of the object files that use those classes. Normally, local copies of certain information (backup copies of inline member functions, debugging information, and the internal tables that implement virtual functions) must be kept in each object file that includes class definitions. You can use this pragma to avoid such duplication. When a header file containing **#pragma interface** is included in a compilation, this auxiliary information will not be generated (unless the main input source file itself uses **#pragma implementation**). Instead, the object files will contain references to be resolved at link time.

#pragma implementation
#pragma implementation "**objects.h**"

Use this pragma in a main input file, when you want full output from included header files to be generated (and made globally visible). The included header file, in turn, should use **#pragma interface**. Backup copies of inline member functions, debugging information, and the internal tables used to implement virtual functions are all generated in implementation files.

If you use **#pragma implementation** with no argument, it applies to an include file with the same basename as your source file; for example, in 'allclass.cc', **#pragma implementation** by itself is equivalent to **#pragma implementation** "**allclass.h**". Use the string argument if you want a single implementation file to include code from multiple header files.

There is no way to split up the contents of a single header file into multiple implementation files.

FILES

file.h C header (preprocessor) file
file.i preprocessed C source file
file.C C++ source file
file.cc C++ source file
file.cxx C++ source file
file.s assembly language file
file.o object file
a.out link edited output
TMPDIR/cc*
temporary files
LIBDIR/cpp
preprocessor
LIBDIR/cc1plus
compiler
LIBDIR/collect
linker front end needed on some machines
LIBDIR/libgcc.a
GCC subroutine library
/lib/crt[01n].o
start-up routine
LIBDIR/ccrt0
additional start-up routine for C++
/lib/libc.a
standard C library, see intro(3)

/usr/include
standard directory for #include files
LIBDIR/include
standard gcc directory for #include files
LIBDIR/g++-include
additional g++ directory for #include

LIBDIR is usually **/usr/local/lib/machine/version**; on INTERIX it is **$OPENNT_ROOT/usr/contrib/{alpha|intel}-pc-opennt/cygnus-2.7.2-970404**.

TMPDIR comes from the environment variable *TMPDIR* (default **$OPENNT_ROOT/usr/tmp** if available, else **$OPENNT_ROOT/tmp**).

SEE ALSO

as, *cpp*, *gcc* *ld*

The “gcc”, “cpp”, “as”, “ld”, and “gdb” entries in *info*.

Using and Porting GNU CC (for version 2.0), Richard M. Stallman; *The C Preprocessor*, Richard M. Stallman; *Debugging with GDB: the GNU Source-Level Debugger*, Richard M. Stallman and Roland H. Pesch; *Using as: the GNU Assembler*, Dean Elsner, Jay Fenlason & friends; *gld: the GNU linker*, Steve Chamberlain and Roland Pesch.

BUGS

For instructions on how to report bugs, see the GCC manual.

COPYING

Copyright (c) 1991, 1992, 1993 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

AUTHORS

See the GNU CC Manual for the contributors to GNU CC.

NAME

gcc, g++ – GNU project C and C++ Compiler

SYNOPSIS

gcc **[***option***|***filename***]**. . .

WARNING

The information in this man page is an extract from the full documentation of the GNU C compiler, and is limited to the meaning of the options.

This man page is not kept up to date except when volunteers want to maintain it. If you find a discrepancy between the man page and the software, please check the Info file, which is the authoritative documentation.

If we find that the things in this man page that are out of date cause significant confusion or complaints, we will stop distributing the man page. The alternative, updating the man page when we update the Info file, is impossible because the rest of the work of maintaining GNU CC leaves us no time for that. The GNU project regards man pages as obsolete and should not let them take time away from other things.

For complete and current documentation, refer to the Info file **gcc** or the manual *Using and Porting GNU CC* (for version 2.0). Both are made from the Texinfo source file **gcc.texinfo**.

DESCRIPTION

The C and C++ compilers are integrated. Both process input files through one or more of four stages: preprocessing, compilation, assembly, and linking. Source filename suffixes identify the source language, but which name you use for the compiler governs default assumptions:

gcc assumes preprocessed (**.i**) files are C and assumes C style linking.
g++ assumes preprocessed ((**.i**) files are C++ and assumes C++ style linking.

Suffixes of source file names indicate the language and kind of processing to be done:

.c C source; preprocess, compile, assemble
.C C++ source; preprocess, compile, assemble
.cc C++ source; preprocess, compile, assemble
.cxx C++ source; preprocess, compile, assemble
.m Objective-C source; preprocess, compile, assemble
.i preprocessed C; compile, assemble
.ii preprocessed C++; compile, assemble
.s Assembler source; assemble
.S Assembler source; preprocess, assemble
.h Preprocessor file; not usually named on command line

Files with other suffixes are passed to the linker. Common cases include:

.o Object file
.a Archive file

Linking is always the last stage unless you use one of the **–c**, **–S**, or **–E** options to avoid it (or unless compilation errors stop the whole process). For the link stage, all **.o** files

corresponding to source files, **–l** libraries, unrecognized filenames (including named **.o** object files and **.a** archives) are passed to the linker in command-line order.

OPTIONS

Options must be separate: **–dr** is quite different from **–d –r**.

Most **–f** and **–W** options have two contrary forms: **–f***name* and **–fno–***name* (or **–W***name* and **–Wno–***name*). Only the non-default forms are shown here.

Here is a summary of all the options, grouped by type. Explanations are in the following sections.

Overall Options

–c
–S
–E
–o *file*
–pipe
–v
–x *language*

Language Options

–ansi
–fall–virtual
–fcond–mismatch
–fdollars–in–identifiers
–fenum–int–equiv
–fexternal–templates
–fno–asm
–fno–builtin
–fhosted
–fno–hosted
–ffreestanding
–fno–freestanding
–fno–strict–prototype
–fsigned–bitfields
–fsigned–char
–fthis–is–variable
–funsigned–bitfields
–funsigned–char
–fwritable–strings
–traditional
–traditional–cpp
–trigraphs

Warning Options

–fsyntax–only
–pedantic
–pedantic–errors
–w
–W
–Wall
–Waggregate–return
–Wcast–align
–Wcast–qual

–Wchar–subscript
–Wcomment
–Wconversion
–Wenum–clash
–Werror
–Wformat
–Wid–clash–*len*
–Wimplicit
–Winline
–Wmain
–Wmissing–prototypes
–Wmissing–declarations
–Wnested–externs
–Wno–import
–Wparentheses
–Wpointer–arith
–Wredundant–decls
–Wreturn–type
–Wshadow
–Wstrict–prototypes
–Wswitch
–Wtemplate–debugging
–Wtraditional
–Wtrigraphs
–Wuninitialized
–Wunused
–Wwrite–strings

Debugging Options

–a
–d*letters*
–fpretend–float
–g
–g*level*
–gcoff
–gxcoff
–gxcoff+
–gdwarf
–gdwarf+
–gstabs
–gstabs+
–ggdb
–p
–pg
–save–temps
–print–file–name=*library*
–print–libgcc–file–name
–print–prog–name=*program*

Optimization Options

–fcaller–saves
–fcse–follow–jumps

–fcse–skip–blocks
–fdelayed–branch
–felide–constructors
–fexpensive–optimizations
–ffast–math
–ffloat–store
–fforce–addr
–fforce–mem
–finline–functions
–fkeep–inline–functions
–fmemoize–lookups
–fno–default–inline
–fno–defer–pop
–fno–function–cse
–fno–inline
–fno–peephole
–fomit–frame–pointer
–frerun–cse–after–loop
–fschedule–insns
–fschedule–insns2
–fstrength–reduce
–fthread–jumps
–funroll–all–loops
–funroll–loops
–O
–O2

Preprocessor Options

–A *assertion*
–C
–dD
–dM
–dN
–D*macro*[**=***defn*]
–E
–H
–idirafter *dir*
–include *file*
–imacros *file*
–iprefix *file*
–iwithprefix *dir*
–M
–MD
–MM
–MMD
–nostdinc
–P
–U *macro*
–undef

Assembler

–Wa,*option*

Linker

–l*library*
–nostartfiles
–nostdlib
–static
–shared
–symbolic
–Xlinker *option*
–Wl,*option*
–u *symbol*

Directory Options

–B*prefix* **–I***dir* **–I– –L***dir*

Target Options

–b *machine*
–V *version*

Configuration Dependent Options

DEC Alpha Options

–mfp–regs
–mno–fp–regs
–mno–soft–float
–msoft–float

SPARC Options

–mepilogue
–mfpu
–mhard–float
–mno–fpu
–mno–epilogue
–msoft–float
–msparclite
–mv8
–msupersparc
–mcypress

Convex Options

–margcount
–mc1
–mc2
–mnoargcount

AMD29K Options

–m29000
–m29050
–mbw
–mdw
–mkernel–registers
–mlarge
–mnbw
–mnodw
–msmall
–mstack–check

–muser–registers

M88K Options

–m88000
–m88100
–m88110
–mbig–pic
–mcheck–zero–division
–mhandle–large–shift
–midentify–revision
–mno–check–zero–division
–mno–ocs–debug–info
–mno–ocs–frame–position
–mno–optimize–arg–area
–mno–serialize–volatile
–mno–underscores
–mocs–debug–info
–mocs–frame–position
–moptimize–arg–area
–mserialize–volatile
–mshort–data–*num*
–msvr3
–msvr4
–mtrap–large–shift
–muse–div–instruction
–mversion–03.00
–mwarn–passed–structs

RS6000 Options

–mfp–in–toc
–mno–fop–in–toc

RT Options

–mcall–lib–mul
–mfp–arg–in–fpregs
–mfp–arg–in–gregs
–mfull–fp–blocks
–mhc–struct–return
–min–line–mul
–mminimum–fp–blocks
–mnohc–struct–return

MIPS Options

–mcpu=*cpu type*
–mips2
–mips3
–mint64
–mlong64
–mlonglong128
–mmips–as
–mgas
–mrnames
–mno–rnames

–mgpopt
–mno–gpopt
–mstats
–mno–stats
–mmemcpy
–mno–memcpy
–mno–mips–tfile
–mmips–tfile
–msoft–float
–mhard–float
–mabicalls
–mno–abicalls
–mhalf–pic
–mno–half–pic
–G *num*
–nocpp

i386 Options

–m486
–mno–486
–msoft–float
–mno–fp–ret–in–387

HPPA Options

–mpa–risc–1–0
–mpa–risc–1–1
–mkernel
–mshared–libs
–mno–shared–libs
–mlong–calls
–mdisable–fpregs
–mdisable–indexing
–mtrailing–colon

i960 Options

–m*cpu-type*
–mnumerics
–msoft–float
–mleaf–procedures
–mno–leaf–procedures
–mtail–call
–mno–tail–call
–mcomplex–addr
–mno–complex–addr
–mcode–align
–mno–code–align
–mic–compat
–mic2.0–compat
–mic3.0–compat
–masm–compat
–mintel–asm
–mstrict–align
–mno–strict–align

–mold–align
–mno–old–align

System V Options

–G
–Qy
–Qn
–YP,*paths*
–Ym,*dir*

Code Generation Options

–fcall–saved–*reg*
–fcall–used–*reg*
–ffixed–*reg*
–finhibit–size–directive
–fnonnull–objects
–fno–common
–fno–ident
–fno–gnu–linker
–fpcc–struct–return
–fpic
–fPIC
–freg–struct–return
–fshared–data
–fshort–enums
–fshort–double
–fvolatile
–fvolatile–global
–fverbose–asm

OVERALL OPTIONS

–x *language*
Specify explicitly the *language* for the following input files (rather than choosing a default based on the file name suffix) . This option applies to all following input files until the next **–x** option. Possible values of *language* are **c**, **objective–c**, **c–header**, **c++**, **cpp–output**, **assembler**, **assembler–with–cpp**.

–x *none* Turn off any specification of a language, so that subsequent files are handled according to their file name suffixes (as they are if **–x** has not been used at all).

If you want only some of the four stages (preprocess, compile, assemble, link), you can use **–x** (or filename suffixes) to tell *gcc* where to start, and one of the options **–c**, **–S**, or **–E** to say where *gcc* is to stop. Note that some combinations (for example, **–x cpp–output –E**) instruct *gcc* to do nothing at all.

–c Compile or assemble the source files, but do not link. The compiler output is an object file corresponding to each source file.

By default, GCC makes the object file name for a source file by replacing the suffix **.c**, **.i**, **.s**, etc., with **.o**. Use **–o** to select another name.

GCC ignores any unrecognized input files (those that do not require compilation or assembly) with the **–c** option.

–S Stop after the stage of compilation proper; do not assemble. The output is an assembler code file for each non-assembler input file specified.

By default, GCC makes the assembler file name for a source file by replacing the suffix **.c**, **.i**, etc., with **.s**. Use **–o** to select another name.

GCC ignores any input files that don't require compilation.

–E Stop after the preprocessing stage; do not run the compiler proper. The output is preprocessed source code, which is sent to the standard output.

GCC ignores input files which don't require preprocessing.

–o *file* Place output in file *file*. This applies regardless to whatever sort of output GCC is producing, whether it be an executable file, an object file, an assembler file or preprocessed C code.

Since only one output file can be specified, it does not make sense to use **–o** when compiling more than one input file, unless you are producing an executable file as output.

If you do not specify **–o**, the default is to put an executable file in **a.out**, the object file for **source**.**suffix** in *source***.o**, its assembler file in *source***.o**, and all preprocessed C source on standard output.

–v Print (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper.

–pipe Use pipes rather than temporary files for communication between the various stages of compilation. This fails to work on some systems where the assembler cannot read from a pipe; but the GNU assembler has no trouble.

LANGUAGE OPTIONS

The following options control the dialect of C that the compiler accepts:

–ansi Support all ANSI standard C programs.

This turns off certain features of GNU C that are incompatible with ANSI C, such as the **asm**, **inline**, and **typeof** keywords, and predefined macros such as **vax** that identify the type of system you are using. It also enables the undesirable and rarely used ANSI trigraph feature, and disallows **$** as part of identifiers.

The alternate keywords **__asm__**, **__extension__**, **__inline__**, and **__typeof__** continue to work despite **–ansi**. You would not want to use them in an ANSI C program, of course, but it is useful to put them in header files that might be included in compilations done with **–ansi**. Alternate predefined macros such as **__unix__** and **__vax__** are also available, with or without **–ansi**.

The **–ansi** option does not cause non-ANSI programs to be rejected gratuitously. For that, **–pedantic** is required in addition to **–ansi**.

The preprocessor predefines a macro **__STRICT_ANSI__** when you use the **–ansi** option. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ANSI standard doesn't call for; this is to avoid interfering with any programs that might use these names for other things.

–fno–asm

Do not recognize **asm**, **inline** or **typeof** as a keyword. These words may then be used as identifiers. You can use **__asm__**, **__inline__** and **__typeof__** instead. **–ansi** implies **–fno–asm**.

–fno–builtin

Don't recognize built-in functions that do not begin with two leading underscores. Currently, the functions affected include *_exit*( ), *abort*( ), *abs*( ), *alloca*( ), *cos*( ), *exit*( ), *fabs*( ), *labs*( ), *memcmp*( ), *memcpy*( ), *sin*( ), *sqrt*( ), *strcmp*( ), *strcpy*( ), and *strlen*( ).

The **–ansi** option prevents *alloca*( ) and *_exit*( ) from being builtin functions.

–fhosted

Compile for a hosted environment; this implies the **–fbuiltin** option, and implies that suspicious declarations of *main*( ) should be warned about.

–ffreestanding

Compile for a freestanding environment; this implies the **–fno-builtin** option, and implies that **main** has no special requirements.

–fno–strict–prototype

Treat a function declaration with no arguments, such as **int foo**();, as C would treat it—as saying nothing about the number of arguments or their types (C++ only). Normally, such a declaration in C++ means that the function **foo** takes no arguments.

–trigraphs

Support ANSI C trigraphs. The **–ansi** option implies **–trigraphs**.

–traditional

Attempt to support some aspects of traditional C compilers. For details, see the GNU C Manual; the duplicate list here has been deleted so that we won't get complaints when it is out of date.

But one note about C++ programs only (not C). **–traditional** has one additional effect for C++: assignment to **this** is permitted. This is the same as the effect of **–fthis–is–variable**.

–traditional–cpp

Attempt to support some aspects of traditional C preprocessors. This includes the items that specifically mention the preprocessor above, but none of the other effects of **–traditional**.

–fdollars–in–identifiers

Permit the use of **$** in identifiers (C++ only). You can also use **–fno–dollars–in–identifiers** to explicitly prohibit use of **$**. (GNU C++ allows **$** by default on some target systems but not others.)

–fenum–int–equiv

Permit implicit conversion of **int** to enumeration types (C++ only). Normally GNU C++ allows conversion of **enum** to **int**, but not the other way around.

–fexternal–templates

Produce smaller code for template declarations, by generating only a single copy of each template function where it is defined (C++ only). To use this option successfully, you must also mark all files that use templates with either

#pragma implementation (the definition) or **#pragma interface** (declarations).

When your code is compiled with **–fexternal–templates**, all template instantiations are external. You must arrange for all necessary instantiations to appear in the implementation file; you can do this with a **typedef** that references each instantiation needed. Conversely, when you compile using the default option **–fno–external–templates**, all template instantiations are explicitly internal.

–fall–virtual

Treat all possible member functions as virtual, implicitly. All member functions (except for constructor functions and **new** or **delete** member operators) are treated as virtual functions of the class where they appear.

This does not mean that all calls to these member functions will be made through the internal table of virtual functions. Under some circumstances, the compiler can determine that a call to a given virtual function can be made directly; in these cases the calls are direct in any case.

–fcond–mismatch

Allow conditional expressions with mismatched types in the second and third arguments. The value of such an expression is void.

–fthis–is–variable

Permit assignment to **this** (C++ only). The incorporation of user-defined free store management into C++ has made assignment to **this** an anachronism. Therefore, by default it is invalid to assign to **this** within a class member function. However, for backwards compatibility, you can make it valid with **–fthis-is-variable**.

–funsigned–char

Let the type **char** be unsigned, like **unsigned char**.

Each kind of machine has a default for what **char** should be. It is either like **unsigned char** by default or like **signed char** by default.

Ideally, a portable program should always use **signed char** or **unsigned char** when it depends on the signedness of an object. But many programs have been written to use plain **char** and expect it to be signed, or expect it to be unsigned, depending on the machines they were written for. This option, and its inverse, let you make such a program work with the opposite default.

The type **char** is always a distinct type from each of **signed char** and **unsigned char**, even though its behavior is always just like one of those two.

–fsigned–char

Let the type **char** be signed, like **signed char**.

Note that this is equivalent to **–fno–unsigned–char**, which is the negative form of **–funsigned–char**. Likewise, **–fno–signed–char** is equivalent to **–funsigned–char**.

–fsigned–bitfields
–funsigned–bitfields

–fno–signed–bitfields
–fno–unsigned–bitfields
These options control whether a bitfield is signed or unsigned, when declared with no explicit **signed** or **unsigned** qualifier. By default, such a bitfield is signed, because this is consistent: the basic integer types such as **int** are signed types.

However, when you specify **–traditional**, bitfields are all unsigned no matter what.

–fwritable–strings
Store string constants in the writable data segment and don't uniquize them. This is for compatibility with old programs which assume they can write into string constants. **–traditional** also has this effect.

Writing into string constants is a very bad idea; constants should be constant.

PREPROCESSOR OPTIONS
These options control the C preprocessor, which is run on each C source file before actual compilation.

If you use the **–E** option, GCC does nothing except preprocessing. Some of these options make sense only together with **–E** because they cause the preprocessor output to be unsuitable for actual compilation.

–include *file*
Process *file* as input before processing the regular input file. In effect, the contents of *file* are compiled first. Any **–D** and **–U** options on the command line are always processed before **–include** *file*, regardless of the order in which they are written. All the **–include** and **–imacros** options are processed in the order in which they are written.

–imacros *file*
Process *file* as input, discarding the resulting output, before processing the regular input file. Because the output generated from *file* is discarded, the only effect of **–imacros** *file* is to make the macros defined in *file* available for use in the main input. The preprocessor evaluates any **–D** and **–U** options on the command line before processing **–imacros** *file*, regardless of the order in which they are written. All the **–include** and **–imacros** options are processed in the order in which they are written.

–idirafter *dir*
Add the directory *dir* to the second include path. The directories on the second include path are searched when a header file is not found in any of the directories in the main include path (the one that **–I** adds to).

–iprefix *prefix*
Specify *prefix* as the prefix for subsequent **–iwithprefix** options.

–iwithprefix *dir*
Add a directory to the second include path. The directory's name is made by concatenating *prefix* and *dir*, where *prefix* was specified previously with **–iprefix**.

–nostdinc
Do not search the standard system directories for header files. Only the directories you have specified with **–I** options (and the current directory, if appropriate) are searched.

By using both **–nostdinc** and **–I–**, you can limit the include-file search file to only those directories you specify explicitly.

–nostdinc++
Do not search for header files in the C++–specific standard directories, but do still search the other standard directories. (This option is used when building **libg++**.)

–undef Do not predefine any nonstandard macros. (Including architecture flags).

–E Run only the C preprocessor. Preprocess all the C source files specified and output the results to standard output or to the specified output file.

–C Tell the preprocessor not to discard comments. Used with the **–E** option.

–P Tell the preprocessor not to generate **#line** commands. Used with the **–E** option.

–M [**–MG**]
Tell the preprocessor to output a rule suitable for *make* describing the dependencies of each object file. For each source file, the preprocessor outputs one *make*-rule whose target is the object file name for that source file and whose dependencies are all the files **#include**'d in it. This rule may be a single line or may be continued with \-newline if it is long. The list of rules is printed on standard output instead of the preprocessed C program.

–M implies **–E**.

–MG says to treat missing header files as generated files and assume they live in the same directory as the source file. It must be specified in addition to **–M**.

–MM [**–MG**]
Like **–M** but the output mentions only the user header files included with **#include** "*file*". System header files included with **#include <***file***>** are omitted.

–MD Like **–M** but the dependency information is written to files with names made by replacing **.o** with **.d** at the end of the output file names. This is in addition to compiling the file as specified—**–MD** does not inhibit ordinary compilation the way **–M** does.

The Mach utility **md** can be used to merge the **.d** files into a single dependency file suitable for using with the *make* command.

–MMD
Like **–MD** except mention only user header files, not system header files.

–H Print the name of each header file used, in addition to other normal activities.

–A *question***(***answer***)**
Assert the answer *answer* for *question*, in case it is tested with a preprocessor conditional such as **#if #***question***(***answer***)**. **–A–** disables the standard assertions that normally describe the target machine.

–D *macro*

Define macro *macro* with the string **1** as its definition. Note that on INTERIX the macro **__OPENNT** is automatially defined.

–D *macro*=*defn*

Define macro *macro* as *defn*. All instances of **–D** on the command line are processed before any **–U** options.

–U*macro*

Undefine macro *macro*. **–U** options are evaluated after all **–D** options, but before any **–include** and **–imacros** options.

–dM Tell the preprocessor to output only a list of the macro definitions that are in effect at the end of preprocessing. Used with the **–E** option.

–dD Tell the preprocessor to pass all macro definitions into the output, in their proper sequence in the rest of the output.

–dN Like **–dD** except that the macro arguments and contents are omitted. Only **#define** *name* is included in the output.

ASSEMBLER OPTION

–Wa, *option*

Pass *option* as an option to the assembler. If *option* contains commas, it is split into multiple options at the commas.

LINKER OPTIONS

These options come into play when the compiler links object files into an executable output file. They are meaningless if the compiler is not doing a link step.

object-file-name

A file name that does not end in a special recognized suffix is considered to name an object file or library. (Object files are distinguished from libraries by the linker according to the file contents.) If GCC does a link step, these object files are used as input to the linker.

–l*library*

Use the library named *library* when linking.

The linker searches a standard list of directories for the library, which is actually a file named **lib***library***.a**. The linker then uses this file as if it had been specified precisely by name.

The directories searched include several standard system directories plus any that you specify with **–L**.

Normally the files found this way are library files—archive files whose members are object files. The linker handles an archive file by scanning through it for members which define symbols that have so far been referenced but not defined. However, if the linker finds an ordinary object file rather than a library, the object file is linked in the usual fashion. The only difference between using an **–l** option and specifying a file name is that **–l** surrounds *library* with **lib** and **.a** and searches several directories.

–lobjc You need this special case of the **–l** option in order to link an Objective C program.

–nostartfiles
Do not use the standard system startup files when linking. The standard libraries are used normally.

–nostdlib
Don't use the standard system libraries and startup files when linking. Only the files you specify will be passed to the linker.

–static On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect.

–shared
Produce a shared object which can then be linked with other objects to form an executable. Only a few systems support this option.

–symbolic
Bind references to global symbols when building a shared object. Warn about any unresolved references (unless overridden by the link editor option **–Xlinker –z –Xlinker defs**).
Only a few systems support this option.

–Xlinker *option*
Pass *option* as an option to the linker. You can use this to supply system-specific linker options which GNU CC does not know how to recognize.

If you want to pass an option that takes an argument, you must use **–Xlinker** twice, once for the option and once for the argument. For example, to pass **–assert definitions**, you must write **–Xlinker –assert –Xlinker definitions**. It does not work to write **–Xlinker** "**–assert definitions**", because this passes the entire string as a single argument, which is not what the linker expects.

–Wl, *option*
Pass *option* as an option to the linker. If *option* contains commas, it is split into multiple options at the commas.

–u *symbol*
Pretend the symbol *symbol* is undefined, to force linking of library modules to define it. You can use **–u** multiple times with different symbols to force loading of additional library modules.

DIRECTORY OPTIONS

These options specify directories to search for header files, for libraries and for parts of the compiler:

–I *dir* Append directory *dir* to the list of directories searched for include files.

–I– Any directories you specify with **–I** options before the **–I–** option are searched only for the case of **#include** "*file*"; they are not searched for **#include** <*file*>.

If additional directories are specified with **–I** options after the **–I–**, these directories are searched for all **#include** directives. (Ordinarily *all* **–I** directories are used this way.)

In addition, the **–I–** option inhibits the use of the current directory (where the current input file came from) as the first search directory for **#include** "*file*".

There is no way to override this effect of **–I–**. With **–I.** you can specify searching the directory which was current when the compiler was invoked. That is not exactly the same as what the preprocessor does by default, but it is often satisfactory.

–I– does not inhibit the use of the standard system directories for header files. Thus, **–I–** and **–nostdinc** are independent.

–L *dir* Add directory *dir* to the list of directories to be searched for **–l**.

–B *prefix*

This option specifies where to find the executables, libraries and data files of the compiler itself.

The compiler driver program runs one or more of the subprograms *cpp'*, *cc1'* (or, for C++, *cc1plus'*), *as* and *ld*. It tries *prefix* as a prefix for each program it tries to run, both with and without *machine*/*version*/.

For each subprogram to be run, the compiler driver first tries the **–B** prefix, if any. If that name is not found, or if **–B** was not specified, the driver tries two standard prefixes, which are **/usr/lib/gcc/** and **/usr/local/lib/gcc-lib/**. If neither of those results in a file name that is found, the compiler driver searches for the unmodified program name, using the directories specified in your *PATH* environment variable.

The run-time support file **libgcc.a** is also searched for using the **–B** prefix, if needed. If it is not found there, the two standard prefixes above are tried, and that is all. The file is left out of the link if it is not found by those means. Most of the time, on most machines, **libgcc.a** is not actually necessary.

You can get a similar result from the environment variable *GCC_EXEC_PREFIX*; if it is defined, its value is used as a prefix in the same way. If both the **–B** option and the *GCC_EXEC_PREFIX* variable are present, the **–B** option is used first and the environment variable value second.

WARNING OPTIONS

Warnings are diagnostic messages that report constructions which are not inherently erroneous but which are risky or suggest there may have been an error.

These options control the amount and kinds of warnings produced by GNU CC:

–fsyntax–only

Check the code for syntax errors, but don't emit any output.

–w Inhibit all warning messages.

–Wno–import

Inhibit warning messages about the use of **#import**.

–pedantic

Issue all the warnings demanded by strict ANSI standard C; reject all programs that use forbidden extensions.

Valid ANSI standard C programs should compile properly with or without this option (though a rare few will require **–ansi**. However, without this option, certain GNU extensions and traditional C features are supported as well. With this option, they are rejected. There is no reason to *use* this option; it exists only to satisfy pedants.

–pedantic does not cause warning messages for use of the alternate keywords whose names begin and end with **__**. Pedantic warnings are also disabled in the expression that follows **__extension__**. However, only system header files should use these escape routes; application programs should avoid them.

–pedantic–errors
Like **–pedantic**, except that errors are produced rather than warnings.

–W Print extra warning messages for these events:

- A nonvolatile automatic variable might be changed by a call to *longjmp*( ). These warnings are possible only in optimizing compilation.

  The compiler sees only the calls to *setjmp*( ). It cannot know where *longjmp*( ) will be called; in fact, a signal handler could call it at any point in the code. As a result, you may get a warning even when there is in fact no problem because *longjmp*( ) cannot in fact be called at the place which would cause a problem.

- A function can return either with or without a value. (Falling off the end of the function body is considered returning without a value.) For example, this function would evoke such a warning:

```
foo (a)
{
  if (a > 0)
    return a;
}
```

  Spurious warnings can occur because GNU CC does not realize that certain functions (including *abort*( ) and *longjmp*( )) will never return.

- An expression-statement or the left-hand side of a comma expression contains no side effects. To suppress the warning, cast the unused expression to void. For example, an expression such as **x[i,j]** will cause a warning, but **x[(void)i,j]** will not.

- An unsigned value is compared against zero with **>** or **<=**.

–Wimplicit
Warn whenever a function or parameter is implicitly declared.

–Wmain
Warn if the *main*( ) function is declared or defined with a suspicious type. Typically, it is a function with external linkage, returning **int**, and taking zero or two arguments.

–Wreturn–type
Warn whenever a function is defined with a return-type that defaults to **int**. Also warn about any **return** statement with no return-value in a function whose return-type is not **void**.

–Wunused
Warn whenever a local variable is unused aside from its declaration, whenever a function is declared static but never defined, and whenever a statement computes a result that is explicitly not used.

–Wswitch
Warn whenever a **switch** statement has an index of enumeral type and lacks a **case** for one or more of the named codes of that enumeration. (The presence of a **default** label prevents this warning.) **case** labels outside the enumeration range also provoke warnings when this option is used.

–Wcomment
Warn whenever a comment-start sequence **/*** appears in a comment.

–Wtrigraphs
Warn if any trigraphs are encountered (assuming they are enabled).

–Wformat
Check calls to *printf*( ) and *scanf*( ), etc., to make sure that the arguments supplied have types appropriate to the format string specified.

–Wchar–subscripts
Warn if an array subscript has type **char**. This is a common cause of error, as programmers often forget that this type is signed on some machines.

–Wuninitialized
An automatic variable is used without first being initialized.

These warnings are possible only in optimizing compilation, because they require data flow information that is computed only when optimizing. If you don't specify **–O**, you simply won't get these warnings.

These warnings occur only for variables that are candidates for register allocation. Therefore, they do not occur for a variable that is declared **volatile**, or whose address is taken, or whose size is other than 1, 2, 4 or 8 bytes. Also, they do not occur for structures, unions or arrays, even when they are in registers.

Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by data flow analysis before the warnings are printed.

These warnings are made optional because GNU CC is not smart enough to see all the reasons why the code might be correct despite appearing to have an error. Here is one example of how this can happen:

```
{
  int x;
  switch (y)
    {
    case 1: x = 1;
      break;
    case 2: x = 4;
      break;
    case 3: x = 5;
    }
  foo (x);
}
```

If the value of **y** is always 1, 2 or 3, then **x** is always initialized, but GNU CC doesn't know this. Here is another common case:

```
{
  int save_y;
  if (change_y) save_y = y, y = new_y;
  ...
  if (change_y) y = save_y;
}
```

This has no bug because **save_y** is used only if it is set.

Some spurious warnings can be avoided if you declare as **volatile** all the functions you use that never return.

–Wparentheses
Warn if parentheses are omitted in certain contexts.

–Wtemplate–debugging
When using templates in a C++ program, warn if debugging is not yet fully available (C++ only).

–Wall
All of the above **–W** options combined. These are all the options which pertain to usage that we recommend avoiding and that we believe is easy to avoid, even in conjunction with macros.

The remaining **–W...** options are not implied by **–Wall** because they warn about constructions that we consider reasonable to use, on occasion, in clean programs.

–Wtraditional
Warn about certain constructs that behave differently in traditional and ANSI C.

- Macro arguments occurring within string constants in the macro body. These would substitute the argument in traditional C, but are part of the constant in ANSI C.
- A function declared external in one block and then used after the end of the block.
- A **switch** statement has an operand of type **long**.

–Wshadow
Warn whenever a local variable shadows another local variable.

–Wid–clash– *len*
Warn whenever two distinct identifiers match in the first *len* characters. This may help you prepare a program that will compile with certain obsolete, brain-damaged compilers.

–Wpointer–arith
Warn about anything that depends on the size of a function type or of **void**. GNU C assigns these types a size of 1, for convenience in calculations with **void *** pointers and pointers to functions.

–Wcast–qual
Warn whenever a pointer is cast so as to remove a type qualifier from the target type. For example, warn if a **const char *** is cast to an ordinary **char ***.

–Wcast–align
Warn whenever a pointer is cast such that the required alignment of the target is increased. For example, warn if a **char *** is cast to an **int *** on machines where integers can only be accessed at two- or four-byte boundaries.

–Wwrite–strings
Give string constants the type **const char[** *length* **]** so that copying the address of one into a non-**const char *** pointer will get a warning. These warnings will help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using **const** in declarations and prototypes. Otherwise, it will just be a nuisance; this is why we did not make **–Wall** request these warnings.

–Wconversion
Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This includes conversions of fixed point to floating and vice versa, and conversions changing the width or signedness of a fixed point argument except when the same as the default promotion.

–Waggregate–return
Warn if any functions that return structures or unions are defined or called. (In languages where you can return an array, this also elicits a warning.)

–Wstrict–prototypes
Warn if a function is declared or defined without specifying the argument types. (An old-style function definition is permitted without a warning if preceded by a declaration which specifies the argument types.)

–Wmissing–prototypes
Warn if a global function is defined without a previous prototype declaration. This warning is issued even if the definition itself provides a prototype. The aim is to detect global functions that fail to be declared in header files.

–Wmissing–declarations
Warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files.

–Wredundant-decls
Warn if anything is declared more than once in the same scope, even in cases where multiple declaration is valid and changes nothing.

–Wnested-externs
Warn if an declaration is encountered within an function.

–Wenum–clash
Warn about conversion between different enumeration types (C++ only).

–Woverloaded–virtual
(C++ only.) In a derived class, the definitions of virtual functions must match the type signature of a virtual function declared in the base class. Use this option to request warnings when a derived class declares a function that may be an erroneous attempt to define a virtual function: that is, warn when a function with the same name as a virtual function in the base class, but with a type signature that doesn't match any virtual functions from the base class.

–Winline
Warn if a function can not be inlined, and either it was declared as inline, or else the **–finline–functions** option was given.

–Werror
Treat warnings as errors; abort compilation after any warning.

DEBUGGING OPTIONS

GNU CC has various special options that are used for debugging either your program or GCC:

–g Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.

On most systems that use stabs format, **–g** enables use of extra debugging information that only GDB can use; this extra information makes debugging work better in GDB but will probably make other debuggers crash or refuse to read the program. If you want to control for certain whether to generate the extra information, use **–gstabs+** ', **–gstabs** ', **–gxcoff+** ', **–gxcoff** ', **–gdwarf+** ', or **–gdwarf** (see below).

Unlike most other C compilers, GNU CC allows you to use **–g** with **–O**. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

The following options are useful when GNU CC is generated with the capability for more than one debugging format.

–ggdb Produce debugging information in the native format (if that is supported), including GDB extensions if at all possible.

–gstabs Produce debugging information in stabs format (if that is supported), without GDB extensions. This is the format used by DBX on most BSD systems.

–gstabs+
Produce debugging information in stabs format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program.

–gcoff Produce debugging information in COFF format (if that is supported). This is the format used by SDB on most System V systems prior to System V Release 4.

–gxcoff Produce debugging information in XCOFF format (if that is supported). This is the format used by the DBX debugger on IBM RS/6000 systems.

–gxcoff+
Produce debugging information in XCOFF format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program.

–gdwarf
Produce debugging information in DWARF format (if that is supported). This is the format used by SDB on most System V Release 4 systems.

–gdwarf+
Produce debugging information in DWARF format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program.

–g*level*
–ggdb*level*
–gstabs*level*
–gcoff*level*
–gxcoff*level*
–gdwarf*level*
Request debugging information and also use *level* to specify how much information. The default level is 2.

Level 1 produces minimal information, enough for making backtraces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, but no information about local variables and no line numbers.

Level 3 includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when you use **–g3**.

–p Generate extra code to write profile information suitable for the analysis program *prof*.

–pg Generate extra code to write profile information suitable for the analysis program *gprof*.

–a Generate extra code to write profile information for basic blocks, which will record the number of times each basic block is executed. This data could be analyzed by a program like *tcov*. Note, however, that the format of the data is not what *tcov* expects. Eventually GNU *gprof* should be extended to process this data.

–ax Generate extra code to read basic block profiling parameters from file 'bb.in' and write profiling results to file 'bb.out'. 'bb.in' contains a list of functions. Whenever a function on the list is entered, profiling is turned on. When the outmost function is left, profiling is turned off. If a function name is prefixed with '-' the function is excluded from profiling. If a function name is not unique it can be disambiguated by writing '/path/filename.d:functionname'. 'bb.out' will list some available filenames. Four function names have a special meaning: '__bb_jumps__' will cause jump frequencies to be written to 'bb.out'. '__bb_trace__' will cause the sequence of basic blocks to be piped into 'gzip' and written to file 'bbtrace.gz'. '__bb_hidecall__' will cause call instructions to be excluded from the trace. '__bb_showret__' will cause return instructions to be included in the trace.

–d *letters*

Says to make debugging dumps during compilation at times specified by *letters*. This is used for debugging the compiler. The file names for most of the dumps are made by appending a word to the source file name (e.g. **foo.c.rtl** or **foo.c.jump**).

–dM Dump all macro definitions, at the end of preprocessing, and write no output.

–dN Dump all macro names, at the end of preprocessing.

–dD Dump all macro definitions, at the end of preprocessing, in addition to normal output.

–dy Dump debugging information during parsing, to standard error.

–dr Dump after RTL generation, to *file***.rtl**.

–dx Just generate RTL for a function instead of compiling it. Usually used with **r**.

–dj Dump after first jump optimization, to *file***jump**.

–ds Dump after CSE (including the jump optimization that sometimes follows CSE), to *file***.cse**.

–dL Dump after loop optimization, to *file***.loop**.

–dt Dump after the second CSE pass (including the jump optimization that sometimes follows CSE), to *file***.cse2**.

–df Dump after flow analysis, to *file***.flow**.

–dc Dump after instruction combination, to *file***.combine**.

–dS Dump after the first instruction scheduling pass, to *file***.sched**.

–dl Dump after local register allocation, to *file***.lreg**.

–dg Dump after global register allocation, to *file***.greg**.

–dR Dump after the second instruction scheduling pass, to *file***.sched2**.

–dJ Dump after last jump optimization, to *file***.jump2**'.

–dd Dump after delayed branch scheduling, to *file***.dbr**.

–dk Dump after conversion from registers to stack, to *file***.stack**'.

–da Produce all the dumps listed above.

–dm Print statistics on memory usage, at the end of the run, to standard error.

–dp Annotate the assembler output with a comment indicating which pattern and alternative was used.

–fpretend–float

When running a cross-compiler, pretend that the target machine uses the same floating point format as the host machine. This causes incorrect output of the actual floating constants, but the actual instruction sequence will probably be the same as GNU CC would make when running on the target machine.

–save–temps
Store the usual temporary intermediate files permanently; place them in the current directory and name them based on the source file. Thus, compiling **foo.c** with **–c**–save–temps would produce files **foo.cpp** and **foo.s**, as well as **foo.o**.

–print–file–name= *library*
Print the full absolute name of the library file *library* that would be used when linking—and do not do anything else. With this option, GNU CC does not compile or link anything; it just prints the file name.

–print–libgcc–file–name
Same as **–print–file–name=libgcc.a**.

–print–prog–name= *program*
Like **–print–file–name**, but searches for a program such as *cpp*.

OPTIMIZATION OPTIONS

These options control various sorts of optimizations:

–O
–O1 Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

Without **–O**, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Without **–O**, only variables declared **register** are allocated in registers. The resulting compiled code is a little worse than produced by PCC without **–O**.

With **–O**, the compiler tries to reduce code size and execution time.

When you specify **–O**, the two options **–fthread–jumps** and **–fdefer–pop** are turned on. On machines that have delay slots, the **–fdelayed–branch** option is turned on. For those machines that can support debugging even without a frame pointer, the **–fomit–frame–pointer** option is turned on. On some machines other flags may also be turned on.

–O2 Optimize even more. Nearly all supported optimizations that do not involve a space-speed tradeoff are performed. Loop unrolling and function inlining are not done, for example. As compared to **–O**, this option increases both compilation time and the performance of the generated code.

–O3 Optimize yet more. This turns on everything **–O2** does, along with also turning on **–finline–functions**.

–O0 Do not optimize.

If you use multiple **–O** options, with or without level numbers, the last such option is the one that is effective.

Options of the form **–f***flag* specify machine-independent flags. Most flags have both positive and negative forms; the negative form of **–ffoo** would be **–fno–foo**. The following list shows only one form—the one which is not the default. You can figure out the

other form by either removing **no–** or adding it.

–ffloat–store

Do not store floating point variables in registers. This prevents undesirable excess precision on machines such as the 68000 where the floating registers (of the 68881) keep more precision than a **double** is supposed to have.

For most programs, the excess precision does only good, but a few programs rely on the precise definition of IEEE floating point. Use **–ffloat–store** for such programs.

–fmemoize–lookups
–fsave–memoized

Use heuristics to compile faster (C++ only). These heuristics are not enabled by default, since they are only effective for certain input files. Other input files compile more slowly.

The first time the compiler must build a call to a member function (or reference to a data member), it must (1) determine whether the class implements member functions of that name; (2) resolve which member function to call (which involves figuring out what sorts of type conversions need to be made); and (3) check the visibility of the member function to the caller. All of this adds up to slower compilation. Normally, the second time a call is made to that member function (or reference to that data member), it must go through the same lengthy process again. This means that code like this

```
cout << "This " << p << " has " << n << " legs.\n";
```

makes six passes through all three steps. By using a software cache, a hit significantly reduces this cost. Unfortunately, using the cache introduces another layer of mechanisms which must be implemented, and so incurs its own overhead. **–fmemoize–lookups** enables the software cache.

Because access privileges (visibility) to members and member functions may differ from one function context to the next, *g++* may need to flush the cache. With the **–fmemoize–lookups** flag, the cache is flushed after every function that is compiled. The **–fsave–memoized** flag enables the same software cache, but when the compiler determines that the context of the last function compiled would yield the same access privileges of the next function to compile, it preserves the cache. This is most helpful when defining many member functions for the same class: with the exception of member functions which are friends of other classes, each member function has exactly the same access privileges as every other, and the cache need not be flushed.

–fno–default–inline

Don't make member functions inline by default merely because they are defined inside the class scope (C++ only).

–fno–defer–pop

Always pop the arguments to each function call as soon as that function returns. For machines which must pop arguments after a function call, the compiler normally lets arguments accumulate on the stack for several function calls and pops them all at once.

–fforce–mem
Force memory operands to be copied into registers before doing arithmetic on them. This may produce better code by making all memory references potential common subexpressions. When they are not common subexpressions, instruction combination should eliminate the separate register-load. I am interested in hearing about the difference this makes.

–fforce–addr
Force memory address constants to be copied into registers before doing arithmetic on them. This may produce better code just as **–fforce–mem** may. I am interested in hearing about the difference this makes.

–fomit–frame–pointer
Don't keep the frame pointer in a register for functions that don't need one. This avoids the instructions to save, set up and restore frame pointers; it also makes an extra register available in many functions. *It also makes debugging impossible on most machines*.

On some machines, such as the Vax, this flag has no effect, because the standard calling sequence automatically handles the frame pointer and nothing is saved by pretending it doesn't exist. The machine-description macro **FRAME_POINTER_REQUIRED** controls whether a target machine supports this flag.

–finline–functions
Integrate all simple functions into their callers. The compiler heuristically decides which functions are simple enough to be worth integrating in this way.

If all calls to a given function are integrated, and the function is declared **static**, then GCC normally does not output the function as assembler code in its own right.

–fcaller–saves
Enable values to be allocated in registers that will be clobbered by function calls, by emitting extra instructions to save and restore the registers around such calls. Such allocation is done only when it seems to result in better code than would otherwise be produced.

This option is enabled by default on certain machines, usually those which have no call-preserved registers to use instead.

–fkeep–inline–functions
Even if all calls to a given function are integrated, and the function is declared **static**, nevertheless output a separate run-time callable version of the function.

–fno–function–cse
Do not put function addresses in registers; make each instruction that calls a constant function contain the function's address explicitly.

This option results in less efficient code, but some strange hacks that alter the assembler output may be confused by the optimizations performed when this option is not used.

–fno–peephole
Disable any machine-specific peephole optimizations.

–ffast-math
This option allows GCC to violate some ANSI or IEEE rules/specifications in the interest of optimizing code for speed. For example, it allows the compiler to assume arguments to the **sqrt** function are non-negative numbers.

This option should never be turned on by any **–O** option since it can result in incorrect output for programs which depend on an exact implementation of IEEE or ANSI rules/specifications for math functions.

The following options control specific optimizations. The **–O2** option turns on all of these optimizations except **–funroll–loops** and **–funroll–all–loops**.

The **–O** option usually turns on the **–fthread–jumps** and **–fdelayed–branch** options, but specific machines may change the default optimizations.

You can use the following flags in the rare cases when fine-tuning of optimizations to be performed is desired.

–fstrength–reduce
Perform the optimizations of loop strength reduction and elimination of iteration variables.

–fthread–jumps
Perform optimizations where we check to see if a jump branches to a location where another comparison subsumed by the first is found. If so, the first branch is redirected to either the destination of the second branch or a point immediately following it, depending on whether the condition is known to be true or false.

–funroll–loops
Perform the optimization of loop unrolling. This is only done for loops whose number of iterations can be determined at compile time or run time.

–funroll–all–loops
Perform the optimization of loop unrolling. This is done for all loops. This usually makes programs run more slowly.

–fcse–follow–jumps
In common subexpression elimination, scan through jump instructions when the target of the jump is not reached by any other path. For example, when CSE encounters an **if** statement with an **else** clause, CSE will follow the jump when the condition tested is false.

–fcse–skip–blocks
This is similar to **–fcse–follow–jumps**, but causes CSE to follow jumps which conditionally skip over blocks. When CSE encounters a simple **if** statement with no else clause, **–fcse–skip–blocks** causes CSE to follow the jump around the body of the **if**.

–frerun–cse–after–loop
Re-run common subexpression elimination after loop optimizations has been performed.

–felide–constructors
Elide constructors when this seems plausible (C++ only). With this flag, GNU C++ initializes **y** directly from the call to **foo** without going through a temporary in the following code:

```
A foo ();
A y = foo ();
```

Without this option, GNU C++ first initializes **y** by calling the appropriate constructor for type **A**; then assigns the result of **foo** to a temporary; and, finally, replaces the initial value of **y** with the temporary.

The default behavior (**–fno–elide–constructors**) is specified by the draft ANSI C++ standard. If your program's constructors have side effects, using **–felide-constructors** can make your program act differently, since some constructor calls may be omitted.

–fexpensive–optimizations
Perform a number of minor optimizations that are relatively expensive.

–fdelayed–branch
If supported for the target machine, attempt to reorder instructions to exploit instruction slots available after delayed branch instructions.

–fschedule–insns
If supported for the target machine, attempt to reorder instructions to eliminate execution stalls due to required data being unavailable. This helps machines that have slow floating point or memory load instructions by allowing other instructions to be issued until the result of the load or floating point instruction is required.

–fschedule–insns2
Similar to **–fschedule–insns**, but requests an additional pass of instruction scheduling after register allocation has been done. This is especially useful on machines with a relatively small number of registers and where memory load instructions take more than one cycle.

TARGET OPTIONS

By default, GNU CC compiles code for the same type of machine that you are using. However, it can also be installed as a cross-compiler, to compile for some other type of machine. In fact, several different configurations of GNU CC, for different target machines, can be installed side by side. Then you specify which one to use with the **–b** option.

In addition, older and newer versions of GNU CC can be installed side by side. One of them (probably the newest) will be the default, but you may sometimes wish to use another.

–b *machine*
The argument *machine* specifies the target machine for compilation. This is useful when you have installed GNU CC as a cross-compiler.

The value to use for *machine* is the same as was specified as the machine type when configuring GNU CC as a cross-compiler. For example, if a cross-compiler was configured with **configure i386v**, meaning to compile for an 80386 running System V, then you would specify **–b i386v** to run that cross compiler.

When you do not specify **–b**, it normally means to compile for the same type of machine that you are using.

–V *version*

The argument *version* specifies which version of GNU CC to run. This is useful when multiple versions are installed. For example, *version* might be **2.0**, meaning to run GNU CC version 2.0.

The default version, when you do not specify **–V**, is controlled by the way GNU CC is installed. Normally, it will be a version that is recommended for general use.

MACHINE DEPENDENT OPTIONS

Each of the target machine types can have its own special options, starting with **–m**, to choose among various hardware models or configurations—for example, 68010 vs 68020, floating coprocessor or none. A single installed version of the compiler can compile for any model or configuration, according to the options specified.

Some configurations of the compiler also support additional special options, usually for command-line compatibility with other compilers on the same platform.

These **–m** switches are supported on the SPARC:

–mfpu, **–mhard–float**

Generate output containing floating point instructions. This is the default.

–mno–fpu, **–msoft–float**

Generate output containing library calls for floating point. *Warning:* there is no GNU floating-point library for SPARC. Normally the facilities of the machine's usual C compiler are used, but this cannot be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

–msoft–float changes the calling convention in the output file; therefore, it is only useful if you compile *all* of a program with this option.

–mno–epilogue, **–mepilogue**

With **–mepilogue** (the default), the compiler always emits code for function exit at the end of each function. Any function exit in the middle of the function (such as a return statement in C) will generate a jump to the exit code at the end of the function.

With **–mno–epilogue**, the compiler tries to emit exit code inline at every function exit.

–mno–v8, **–mv8**, **msparclite**

These three options select variations on the SPARC architecture.

By default (unless specifically configured for the Fujitsu SPARClite), GCC generates code for the v7 variant of the SPARC architecture.

–mv8 will give you SPARC v8 code. The only difference from v7 code is that the compiler emits the integer multiply and integer divide instructions which exist in SPARC v8 but not in SPARC v7.

–msparclite will give you SPARClite code. This adds the integer multiply, integer divide step and scan (ffs) instructions which exist in SPARClite but not in SPARC v7.

–mcypress, **–msupersparc**
These two options select the processor for which the code is optimised.

With **–mcypress** (the default), the compiler optimises code for the Cypress CY7C602 chip, as used in the SparcStation / SparcServer 3xx series. This is also appropriate for the older SparcStation 1, 2, IPX etc.

With **–msupersparc** the compiler optimises code for the SuperSparc cpu, as used in the SparcStation 10, 1000 and 2000 series. This flag also enables use of the full SPARC v8 instruction set.

These **–m** options are defined for the Convex:

–mc1 Generate output for a C1. This is the default when the compiler is configured for a C1.

–mc2 Generate output for a C2. This is the default when the compiler is configured for a C2.

–margcount
Generate code which puts an argument count in the word preceding each argument list. Some nonportable Convex and Vax programs need this word. (Debuggers don't, except for functions with variable-length argument lists; this info is in the symbol table.)

–mnoargcount
Omit the argument count word. This is the default if you use the unmodified sources.

These **–m** options are defined for the AMD Am29000:

–mdw Generate code that assumes the DW bit is set, i.e., that byte and halfword operations are directly supported by the hardware. This is the default.

–mnodw
Generate code that assumes the DW bit is not set.

–mbw Generate code that assumes the system supports byte and halfword write operations. This is the default.

–mnbw Generate code that assumes the systems does not support byte and halfword write operations. This implies **–mnodw**.

–msmall
Use a small memory model that assumes that all function addresses are either within a single 256 KB segment or at an absolute address of less than 256K. This allows the **call** instruction to be used instead of a **const**, **consth**, **calli** sequence.

–mlarge Do not assume that the **call** instruction can be used; this is the default.

–m29050
Generate code for the Am29050.

–m29000
Generate code for the Am29000. This is the default.

–mkernel–registers
Generate references to registers **gr64-gr95** instead of **gr96-gr127**. This option can be used when compiling kernel code that wants a set of global registers disjoint from that used by user-mode code.

Note that when this option is used, register names in **–f** flags must use the normal, user-mode, names.

–muser–registers
Use the normal set of global registers, **gr96-gr127**. This is the default.

–mstack–check
Insert a call to **__msp_check** after each stack adjustment. This is often used for kernel code.

These **–m** options are defined for Motorola 88K architectures:

–m88000
Generate code that works well on both the m88100 and the m88110.

–m88100
Generate code that works best for the m88100, but that also runs on the m88110.

–m88110
Generate code that works best for the m88110, and may not run on the m88100.

–midentify–revision
Include an **ident** directive in the assembler output recording the source file name, compiler name and version, timestamp, and compilation flags used.

–mno–underscores
In assembler output, emit symbol names without adding an underscore character at the beginning of each name. The default is to use an underscore as prefix on each name.

–mno–check–zero–division
–mcheck–zero–division
Early models of the 88K architecture had problems with division by zero; in particular, many of them didn't trap. Use these options to avoid including (or to include explicitly) additional code to detect division by zero and signal an exception. All GCC configurations for the 88K use ' **–mcheck–zero–division** by default.

–mocs–debug–info
–mno–ocs–debug–info
Include (or omit) additional debugging information (about registers used in each stack frame) as specified in the 88Open Object Compatibility Standard, OCS. This extra information is not needed by GDB. The default for DG/UX, SVr4, and Delta 88 SVr3.2 is to include this information; other 88k configurations omit this information by default.

–mocs–frame–position
–mno–ocs–frame–position
Force (or do not require) register values to be stored in a particular place in stack frames, as specified in OCS. The DG/UX, Delta88 SVr3.2, and BCS configurations use **–mocs–frame–position**; other 88k configurations have the

default **–mno–ocs–frame–position**.

–moptimize–arg–area
–mno–optimize–arg–area
Control how to store function arguments in stack frames.
–moptimize–arg–area saves space, but may break some debuggers (not GDB).
–mno–optimize–arg–area conforms better to standards. By default GCC does not optimize the argument area.

–mshort–data– *num*
Generate smaller data references by making them relative to **r0**, which allows loading a value using a single instruction (rather than the usual two). You control which data references are affected by specifying *num* with this option. For example, if you specify **–mshort–data–512**, then the data references affected are those involving displacements of less than 512 bytes. **–mshort–data–***num* is not effective for *num* greater than 64K.

–mserialize-volatile
–mno-serialize-volatile
Do, or do not, generate code to guarantee sequential consistency of volatile memory references.

GNU CC always guarantees consistency by default, for the preferred processor submodel. How this is done depends on the submodel.

The m88100 processor does not reorder memory references and so always provides sequential consistency. If you use **–m88100**, GNU CC does not generate any special instructions for sequential consistency.

The order of memory references made by the m88110 processor does not always match the order of the instructions requesting those references. In particular, a load instruction may execute before a preceding store instruction. Such reordering violates sequential consistency of volatile memory references, when there are multiple processors. When you use **–m88000** or **–m88110**, GNU CC generates special instructions when appropriate, to force execution in the proper order.

The extra code generated to guarantee consistency may affect the performance of your application. If you know that you can safely forgo this guarantee, you may use the option **–mno-serialize-volatile**.

If you use the **–m88100** option but require sequential consistency when running on the m88110 processor, you should use **–mserialize-volatile**.

–msvr4
–msvr3
Turn on (**–msvr4**) or off (**–msvr3**) compiler extensions related to System V release 4 (SVr4). This controls the following:

- Which variant of the assembler syntax to emit (which you can select independently using **–mversion–03.00**).
- **–msvr4** makes the C preprocessor recognize **#pragma weak**
- **–msvr4** makes GCC issue additional declaration directives used in SVr4.

–msvr3 is the default for all m88K configurations except the SVr4 configuration.

–mtrap–large–shift
–mhandle–large–shift
Include code to detect bit-shifts of more than 31 bits; respectively, trap such shifts or emit code to handle them properly. By default GCC makes no special provision for large bit shifts.

–muse–div–instruction
Very early models of the 88K architecture didn't have a divide instruction, so GCC avoids that instruction by default. Use this option to specify that it's safe to use the divide instruction.

–mversion–03.00
In the DG/UX configuration, there are two flavors of SVr4. This option modifies **–msvr4** to select whether the hybrid-COFF or real-ELF flavor is used. All other configurations ignore this option.

–mwarn–passed–structs
Warn when a function passes a struct as an argument or result. Structure-passing conventions have changed during the evolution of the C language, and are often the source of portability problems. By default, GCC issues no such warning.

These options are defined for the IBM RS6000:

–mfp–in–toc
–mno–fp–in–toc
Control whether or not floating-point constants go in the Table of Contents (TOC), a table of all global variable and function addresses. By default GCC puts floating-point constants there; if the TOC overflows, **–mno–fp–in–toc** will reduce the size of the TOC, which may avoid the overflow.

These **–m** options are defined for the IBM RT PC:

–min–line–mul
Use an in-line code sequence for integer multiplies. This is the default.

–mcall–lib–mul
Call **lmul$$** for integer multiples.

–mfull–fp–blocks
Generate full-size floating point data blocks, including the minimum amount of scratch space recommended by IBM. This is the default.

–mminimum–fp–blocks
Do not include extra scratch space in floating point data blocks. This results in smaller code, but slower execution, since scratch space must be allocated dynamically.

–mfp–arg–in–fpregs
Use a calling sequence incompatible with the IBM calling convention in which floating point arguments are passed in floating point registers. Note that **varargs.h** and **stdargs.h** will not work with floating point operands if this option is specified.

–mfp–arg–in–gregs
Use the normal calling convention for floating point arguments. This is the default.

–mhc–struct–return
Return structures of more than one word in memory, rather than in a register. This provides compatibility with the MetaWare HighC (hc) compiler. Use **–fpcc–struct–return** for compatibility with the Portable C Compiler (pcc).

–mnohc–struct–return
Return some structures of more than one word in registers, when convenient. This is the default. For compatibility with the IBM-supplied compilers, use either **–fpcc–struct–return** or **–mhc–struct–return**.

These **–m** options are defined for the MIPS family of computers:

–mcpu= *cpu-type*
Assume the defaults for the machine type *cpu-type* when scheduling instructions. The default *cpu-type* is **default**, which picks the longest cycles times for any of the machines, in order that the code run at reasonable rates on all MIPS cpu's. Other choices for *cpu-type* are **r2000**, **r3000**, **r4000**, and **r6000**. While picking a specific *cpu-type* will schedule things appropriately for that particular chip, the compiler will not generate any code that does not meet level 1 of the MIPS ISA (instruction set architecture) without the **–mips2** or **–mips3** switches being used.

–mips2
Issue instructions from level 2 of the MIPS ISA (branch likely, square root instructions). The **–mcpu=r4000** or **–mcpu=r6000** switch must be used in conjunction with **–mips2**.

–mips3
Issue instructions from level 3 of the MIPS ISA (64 bit instructions). The **–mcpu=r4000** switch must be used in conjunction with **–mips2**.

–mint64
–mlong64
–mlonglong128
These options don't work at present.

–mmips–as
Generate code for the MIPS assembler, and invoke **mips–tfile** to add normal debug information. This is the default for all platforms except for the OSF/1 reference platform, using the OSF/rose object format. If any of the **–ggdb**, **–gstabs**, or **–gstabs+** switches are used, the **mips–tfile** program will encapsulate the stabs within MIPS ECOFF.

–mgas
Generate code for the GNU assembler. This is the default on the OSF/1 reference platform, using the OSF/rose object format.

–mrnames
–mno–rnames
The **–mrnames** switch says to output code using the MIPS software names for the registers, instead of the hardware names (ie, **a0** instead of **$4**). The GNU assembler does not support the **–mrnames** switch, and the MIPS assembler will be instructed to run the MIPS C preprocessor over the source file. The

–mno–rnames switch is default.

–mgpopt
–mno–gpopt
The **–mgpopt** switch says to write all of the data declarations before the instructions in the text section, to all the MIPS assembler to generate one word memory references instead of using two words for short global or static data items. This is on by default if optimization is selected.

–mstats
–mno–stats
For each non-inline function processed, the **–mstats** switch causes the compiler to emit one line to the standard error file to print statistics about the program (number of registers saved, stack size, etc.).

–mmemcpy
–mno–memcpy
The **–mmemcpy** switch makes all block moves call the appropriate string function (**memcpy** or **bcopy**) instead of possibly generating inline code.

–mmips–tfile
–mno–mips–tfile
The **–mno–mips–tfile** switch causes the compiler not postprocess the object file with the **mips–tfile** program, after the MIPS assembler has generated it to add debug support. If **mips–tfile** is not run, then no local variables will be available to the debugger. In addition, **stage2** and **stage3** objects will have the temporary file names passed to the assembler embedded in the object file, which means the objects will not compare the same.

–msoft–float
Generate output containing library calls for floating point. *WARNING: the requisite libraries are not part of GNU CC.* Normally the facilities of the machine's usual C compiler are used, but this can't be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

–mhard–float
Generate output containing floating point instructions. This is the default if you use the unmodified sources.

–mfp64
Assume that the **FR** bit in the status word is on, and that there are 32 64-bit floating point registers, instead of 32 32-bit floating point registers. You must also specify the **–mcpu=r4000** and **–mips3** switches.

–mfp32
Assume that there are 32 32-bit floating point registers. This is the default.

–mabicalls
–mno–abicalls
Emit (or do not emit) the **.abicalls**, **.cpload**, and **.cprestore** pseudo operations that some System V.4 ports use for position independent code.

–mhalf–pic
–mno–half–pic
The **–mhalf–pic** switch says to put pointers to extern references into the data section and load them up, rather than put the references in the text section. This option does not work at present.

–G *num*
Put global and static items less than or equal to *num* bytes into the small data or bss sections instead of the normal data or bss section. This allows the assembler to emit one word memory reference instructions based on the global pointer (**gp** or **$28**), instead of the normal two words used. By default, *num* is 8 when the MIPS assembler is used, and 0 when the GNU assembler is used. The **–G** *num* switch is also passed to the assembler and linker. All modules should be compiled with the same **–G** *num* value.

–nocpp
Tell the MIPS assembler to not run it's preprocessor over user assembler files (with a **.s** suffix) when assembling them.

These **–m** options are defined for the Intel 80386 family of computers: **–m486**

–mno–486
Control whether or not code is optimized for a 486 instead of an 386. Code generated for a 486 will run on a 386 and vice versa.

–msoft–float
Generate output containing library calls for floating point. *Warning: the requisite libraries are not part of GNU CC.* Normally the facilities of the machine's usual C compiler are used, but this can't be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

On machines where a function returns floating point results in the 80387 register stack, some floating point opcodes may be emitted even if **–msoft-float** is used.

–mno-fp-ret-in-387
Do not use the FPU registers for return values of functions.

The usual calling convention has functions return values of types **float** and **double** in an FPU register, even if there is no FPU. The idea is that the operating system should emulate an FPU.

The option **–mno-fp-ret-in-387** causes such values to be returned in ordinary CPU registers instead.

These **–m** options are defined for the HPPA family of computers:

–mpa-risc-1-0
Generate code for a PA 1.0 processor.

–mpa-risc-1-1
Generate code for a PA 1.1 processor.

–mkernel
Generate code which is suitable for use in kernels. Specifically, avoid **add** instructions in which one of the arguments is the DP register; generate **addil** instructions instead. This avoids a rather serious bug in the HP-UX linker.

–mshared-libs
Generate code that can be linked against HP-UX shared libraries. This option is not fully function yet, and is not on by default for any PA target. Using this option can cause incorrect code to be generated by the compiler.

–mno-shared-libs
Don't generate code that will be linked against shared libraries. This is the default for all PA targets.

–mlong-calls
Generate code which allows calls to functions greater than 256K away from the caller when the caller and callee are in the same source file. Do not turn this option on unless code refuses to link with branch out of range errors from the linker.

–mdisable-fpregs
Prevent floating point registers from being used in any manner. This is necessary for compiling kernels which perform lazy context switching of floating point registers. If you use this option and attempt to perform floating point operations, the compiler will abort.

–mdisable-indexing
Prevent the compiler from using indexing address modes. This avoids some rather obscure problems when compiling MIG generated code under MACH.

–mtrailing-colon
Add a colon to the end of label definitions (for ELF assemblers).

These **–m** options are defined for the Intel 80960 family of computers:

–m *cpu-type*
Assume the defaults for the machine type *cpu-type* for instruction and addressing-mode availability and alignment. The default *cpu-type* is **kb**; other choices are **ka**, **mc**, **ca**, **cf**, **sa**, and **sb**.

–mnumerics
–msoft–float
The **–mnumerics** option indicates that the processor does support floating-point instructions. The **–msoft–float** option indicates that floating-point support should not be assumed.

–mleaf–procedures
–mno–leaf–procedures
Do (or do not) attempt to alter leaf procedures to be callable with the *bal* instruction as well as *call*. This will result in more efficient code for explicit calls when the *bal* instruction can be substituted by the assembler or linker, but less efficient code in other cases, such as calls via function pointers, or using a linker that doesn't support this optimization.

–mtail–call
–mno–tail–call
Do (or do not) make additional attempts (beyond those of the machine-independent portions of the compiler) to optimize tail-recursive calls into branches. You may not want to do this because the detection of cases where this is not valid is not totally complete. The default is **–mno–tail–call**.

–mcomplex–addr
–mno–complex–addr
Assume (or do not assume) that the use of a complex addressing mode is a win on this implementation of the i960. Complex addressing modes may not be worthwhile on the K-series, but they definitely are on the C-series. The default is currently **–mcomplex–addr** for all processors except the CB and CC.

–mcode–align
–mno–code–align
Align code to 8-byte boundaries for faster fetching (or don't bother). Currently turned on by default for C-series implementations only.

–mic–compat
–mic2.0–compat
–mic3.0–compat
Enable compatibility with iC960 v2.0 or v3.0.

–masm–compat
–mintel–asm
Enable compatibility with the iC960 assembler.

–mstrict–align
–mno–strict–align
Do not permit (do permit) unaligned accesses.

–mold–align
Enable structure-alignment compatibility with Intel's gcc release version 1.3 (based on gcc 1.37). Currently this is buggy in that **#pragma** align**1** is always assumed as well, and cannot be turned off.

These **–m** options are defined for the DEC Alpha implementations:

–mno-soft-float
–msoft-float
Use (do not use) the hardware floating-point instructions for floating-point operations. When **–msoft-float** is specified, functions in **libgcc1.c** will be used to perform floating-point operations. Unless they are replaced by routines that emulate the floating-point operations, or compiled in such a way as to call such emulations routines, these routines will issue floating-point operations. If you are compiling for an Alpha without floating-point operations, you must ensure that the library is built so as not to call them.

Note that Alpha implementations without floating-point operations are required to have floating-point registers.

–mfp-reg
–mno-fp-regs
Generate code that uses (does not use) the floating-point register set. **–mno-fp-regs** implies **–msoft-float**. If the floating-point register set is not used, floating point operands are passed in integer registers as if they were integers and floating-point results are passed in $0 instead of $f0. This is a non-standard calling sequence, so any function with a floating-point argument or return value called by code compiled with **–mno-fp-regs** must also be compiled with that option.

A typical use of this option is building a kernel that does not use, and hence need not save and restore, any floating-point registers.

These additional options are available on System V Release 4 for compatibility with other compilers on those systems:

–G On SVr4 systems, *gcc* accepts the option **–G** (and passes it to the system linker), for compatibility with other compilers. However, we suggest you use **–symbolic** or **–shared** as appropriate, instead of supplying linker options on the *gcc* command line.

–Qy

Identify the versions of each tool used by the compiler, in a **.ident** assembler directive in the output.

–Qn

Refrain from adding **.ident** directives to the output file (this is the default).

–YP, *dirs*

Search the directories *dirs*, and no others, for libraries specified with **–l**. You can separate directory entries in *dirs* from one another with colons.

–Ym, *dir*

Look in the directory *dir* to find the M4 preprocessor. The assembler uses this option.

CODE GENERATION OPTIONS

These machine-independent options control the interface conventions used in code generation.

Most of them begin with **–f**. These options have both positive and negative forms; the negative form of **–ffoo** would be **–fno–foo**. In the table below, only one of the forms is listed—the one which is not the default. You can figure out the other form by either removing **no–** or adding it.

–fnonnull–objects

Assume that objects reached through references are not null (C++ only).

Normally, GNU C++ makes conservative assumptions about objects reached through references. For example, the compiler must check that **a** is not null in code like the following:

```
obj &a = g ();
a.f (2);
```

Checking that references of this sort have non-null values requires extra code, however, and it is unnecessary for many programs. You can use **–fnonnull-objects** to omit the checks for null, if your program doesn't require checking.

–fpcc–struct–return

Use the same convention for returning **struct** and **union** values that is used by the usual C compiler on your system. This convention is less efficient for small structures, and on many machines it fails to be reentrant; but it has the advantage of allowing intercallability between GCC-compiled code and PCC-compiled code.

–freg–struct–return

Use the convention that **struct** and **union** values are returned in registers when possible. This is more efficient for small structures than **–fpcc–struct–return**.

If you specify neither **–fpcc–struct–return** nor **–freg–struct–return**, GNU CC defaults to whichever convention is standard for the target. If there is no standard convention, GNU CC defaults to **–fpcc–struct–return**.

–fshort–enums

Allocate to an **enum** type only as many bytes as it needs for the declared range of possible values. Specifically, the **enum** type will be equivalent to the smallest integer type which has enough room.

–fshort–double

Use the same size for **double** as for **float**.

–fshared–data

Requests that the data and non-**const** variables of this compilation be shared data rather than private data. The distinction makes sense only on certain operating systems, where shared data is shared between processes running the same program, while private data exists in one copy per process.

–fno–common

Allocate even uninitialized global variables in the bss section of the object file, rather than generating them as common blocks. This has the effect that if the same variable is declared (without **extern**) in two different compilations, you will get an error when you link them. The only reason this might be useful is if you wish to verify that the program will work on other systems which always work this way.

–fno–ident

Ignore the **#ident** directive.

–fno–gnu–linker

Do not output global initializations (such as C++ constructors and destructors) in the form used by the GNU linker (on systems where the GNU linker is the standard method of handling them). Use this option when you want to use a non-GNU linker, which also requires using the *collect2* program to make sure the system linker includes constructors and destructors. (*collect2* is included in the GNU CC distribution.) For systems which *must* use *collect2*, the compiler driver *gcc* is configured to do this automatically.

–finhibit-size-directive

Don't output a **.size** assembler directive, or anything else that would cause trouble if the function is split in the middle, and the two halves are placed at locations far apart in memory. This option is used when compiling **crtstuff.c**; you should not need to use it for anything else.

–fverbose-asm

Put extra commentary information in the generated assembly code to make it more readable. This option is generally only of use to those who actually need to read the generated assembly code (perhaps while debugging the compiler itself).

–fvolatile
Consider all memory references through pointers to be volatile.

–fvolatile–global
Consider all memory references to extern and global data items to be volatile.

–fpic If supported for the target machines, generate position-independent code, suitable for use in a shared library.

–fPIC If supported for the target machine, emit position-independent code, suitable for dynamic linking, even if branches need large displacements.

–ffixed–*reg*
Treat the register named *reg* as a fixed register; generated code should never refer to it (except perhaps as a stack pointer, frame pointer or in some other fixed role).

reg must be the name of a register. The register names accepted are machine-specific and are defined in the **REGISTER_NAMES** macro in the machine description macro file.

This flag does not have a negative form, because it specifies a three-way choice.

–fcall–used–*reg*
Treat the register named *reg* as an allocatable register that is clobbered by function calls. It may be allocated for temporaries or variables that do not live across a call. Functions compiled this way will not save and restore the register *reg*.

Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

This flag does not have a negative form, because it specifies a three-way choice.

–fcall–saved–*reg*
Treat the register named *reg* as an allocatable register saved by functions. It may be allocated even for temporaries or variables that live across a call. Functions compiled this way will save and restore the register *reg* if they use it.

Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

A different sort of disaster will result from the use of this flag for a register in which function values may be returned.

This flag does not have a negative form, because it specifies a three-way choice.

PRAGMAS

Two **#pragma** directives are supported for GNU C++, to permit using the same header file for two purposes: as a definition of interfaces to a given object class, and as the full definition of the contents of that object class.

#pragma interface
(C++ only.) Use this directive in header files that define object classes, to save space in most of the object files that use those classes. Normally, local copies of certain information (backup copies of inline member functions, debugging information, and the internal tables that implement virtual functions) must be

kept in each object file that includes class definitions. You can use this pragma to avoid such duplication. When a header file containing **#pragma interface** is included in a compilation, this auxiliary information will not be generated (unless the main input source file itself uses **#pragma implementation**'). Instead, the object files will contain references to be resolved at link time.

#pragma implementation

#pragma implementation *objects*.**h**

(C++ only.) Use this pragma in a main input file, when you want full output from included header files to be generated (and made globally visible). The included header file, in turn, should use **#pragma interface**. Backup copies of inline member functions, debugging information, and the internal tables used to implement virtual functions are all generated in implementation files.

If you use **#pragma implementation** with no argument, it applies to an include file with the same basename as your source file; for example, in **allclass.cc**, **#pragma implementation** by itself is equivalent to **#pragma implementation** "**allclass.h**". Use the string argument if you want a single implementation file to include code from multiple header files.

There is no way to split up the contents of a single header file into multiple implementation files.

FILES

file.c C source file
file.h C header (preprocessor) file
file.i preprocessed C source file
file.C C++ source file
file.cc C++ source file
file.cxx C++ source file
file.m Objective-C source file
file.s assembly language file
file.o object file
a.out link edited output
$TMPDIR/cc*
temporary files
$LIBDIR/cpp
preprocessor
$LIBDIR/cc1
compiler for C
$LIBDIR/cc1plus
compiler for C++
$LIBDIR/collect
linker front end needed on some machines
$LIBDIR/libgcc.a
GCC subroutine library
/lib/crt[01n].o
start-up routine
$LIBDIR/ccrt0
additional start-up routine for C++
/lib/libc.a
standard C library, see *intro*()

$OPENNT_ROOT/usr/include
standard directory for **#include** files
$LIBDIR/include
standard gcc directory for **#include** files
$LIBDIR/g++–include
additional g++ directory for **#include**

LIBDIR is usually **/usr/local/lib/***machine/version*. For INTERIX, it is **$OPENNT_ROOT/usr/contrib/***machine/version*; *machine* is "alpha-pc-opennt" or "i386-pc-opennt".
TMPDIR comes from the environment variable *TMPDIR* (default **$OPENNT_ROOT/usr/tmp** if available, else **$OPENNT_ROOT/tmp**).

SEE ALSO

cpp, *as*, *ld*, *gdb*, *adb*, *dbx*, *sdb*

"gcc", "cpp", "as", "ld", and "gdb" entries in *info*.

Using and Porting GNU CC (for version 2.0), Richard M. Stallman; *The C Preprocessor*, Richard M. Stallman; *Debugging with GDB: the GNU Source-Level Debugger*, Richard M. Stallman and Roland H. Pesch; *Using as: the GNU Assembler*, Dean Elsner, Jay Fenlason & friends; *ld: the GNU linker*, Steve Chamberlain and Roland Pesch.

BUGS

For instructions on reporting bugs, see the GCC manual.

COPYING

Copyright © 1991, 1992, 1993 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

AUTHORS

See the GNU CC Manual for the contributors to GNU CC.

NAME

getconf – get configuration values

SYNOPSIS

```
getconf system_var
getconf path_var pathname
getconf –a
```

DESCRIPTION

The *getconf* utility writes to standard output the value of *system_var* or the value of *path_var* based on the pathname provided as *pathname*.

The operands supported for *system_var* and *path_var* can be found in POSIX.1 and POSIX.2.

If the operand is valid, but the variable is undefined for the system, *getconf* writes the string `undefined` to standard output.

This implementation of *getconf* supports an option (an extension from the standard):

–a List the known system variables. Names that require a pathname are given the pathname ".".

The following system variables are supported for *system_var*:

_SC_2_C_BIND
: Whether system provides POSIX.2 C language bindings.

_SC_2_C_DEV
: Whether system supports the POSIX.2 C-Language Development Utilities Option.

_SC_2_CHAR_TERM
: Whether system supports at least one terminal type capable of supporting all operations described in POSIX.2.

_SC_2_FORT_RUN
: Whether system supports POSIX.2 FORTRAN Runtime Utilities Option.

_SC_2_FORT_DEV
: Whether system supports POSIX.2 FORTRAN Development Utilities Option.

_SC_2_LOCALEDEF
: Whether system supports the creation of locales.

_SC_2_SW_DEV
: Whether system supports POSIX.2 Software Development Portability Utilities Option.

_SC_2_UPE
: Whether system supports POSIX.2 User Portability Utilities Option

_SC_2_VERSION
: Version of ISO / IEC 9945 (POSIX 1003.2) with which the system attempts to comply.

_SC_ARG_MAX
: Maximum bytes of argument to *exec*( ).

_SC_BC_BASE_MAX
: Maximum **obase** value allowed by *bc*.

_SC_BC_DIM_MAX
: Maximum number of elements in a array

_SC_BC_SCALE_MAX

Maximum **scale** value allowed by *bc*.

_SC_BC_STRING_MAX
: Maximum length of a string constant in *bc*.

_SC_CHILD_MAX
: Maximum number of simultaneous processes per user ID.

_SC_CLK_TCK
: Number of micro-seconds per hz tick.

_SC_COLL_WEIGHTS_MAX
: In a locale definition file, the maximum number of weights that can be assigned to the LC_COLLATE **order** keyword.

_SC_EXPR_NEST_MAX
: Maximum number of expressions that can be nested within parentheses by *expr*.

_SC_JOB_CONTROL
: Return 1 if job control is available on this system, otherwise –1.

_SC_LINE_MAX
: Maximum length in bytes, including trailing newline, of an input line for a text-processing utility

_SC_NGROUPS_MAX
: Maximum number of supplemental groups.

_SC_OPEN_MAX
: Maximum number of open files per user ID.

_SC_PAGE_SIZE
: The virtual memory page size.

_SC_PAGESIZE
: The virtual memory page size.

_SC_RE_DUP_MAX
: Maximum number of repetitions of a BRE allowed in **\{***m***,***n***\}** notation

_SC_SAVED_IDS
: Returns 1 if saved set-group and saved set-user ID is available, otherwise –1.

_SC_STREAM_MAX
: Minimum maximum number of streams that a process may have open at any one time.

_SC_TZNAME_MAX
: Minimum maximum number of types supported for the name of a timezone.

_SC_VERSION
: The version of ISO / IEC 9945 (POSIX 1003.1) with which the system attempts to comply.

The following path variables are supported for *path_var*:

_PC_lINK_MAX
: The maximum file link count.

_PC_MAX_CANON
: The maximum number of bytes in terminal canonical input line.

_PC_MAX_INPUT
: The minimum maximum number of bytes for which space is available in a terminal input queue.

_PC_NAME_MAX
: The maximum number of bytes in a file name.

_PC_PATH_MAX
: The maximum number of bytes in a pathname.

_PC_PIPE_BUF
: The maximum number of bytes which will be written atomically to a pipe.

_PC_CHOWN_RESTRICTED
: Return 1 if appropriate privileges are required for the *chown*( ) system call, otherwise 0.

_PC_NO_TRUNC
: Return 1 if file names longer than {KERN_NAME_MAX} are truncated.

_PC_VDISABLE
: Returns the terminal character disabling value.

DIAGNOSTICS

The *getconf* utility exits 0 if the operand is valid, and >0 if an error occurs.

EXAMPLES

To determine the number of child processes supported on INTERIX:

```
$ getconf CHILD_MAX
28
```

To determine the minimum number of child processes mandated by POSIX.1:

```
$ getconf _POSIX_CHILD_MAX
6
```

SEE ALSO

pathconf(), *sysconf*()

NAME

getopts – parse utility options

SYNOPSIS

`getopts` *`optstring var`* **[***`arg ...`***]**

DESCRIPTION

Getopts is used by shell procedures to parse the specified arguments (or positional parameters, if no arguments are given) and to check for legal options. The argument *optstring* contains the option letters that *getopts* is to recognize. If a letter is followed by a colon, the option is expected to have an argument. Arguments containing options must all start with either a **–** or a **+**, options that do not take arguments may be grouped in a single argument. If an option takes an argument and the option character is not the last character of the argument it is found in, the remainder of the argument is taken to be the option's argument, otherwise, the next argument is the option's argument.

Each time *getopts* is invoked, it places the next option in the shell parameter *name* and the index of the next argument to be processed in the shell parameter *OPTIND*. If the option was introduced with a **+**, the option placed in *name* is prefixed with a **+**. When an option requires an argument, *getopts* places it in the shell parameter *OPTARG*. When an illegal option or a missing option argument is encountered a question mark or a colon is placed in *name* (indicating an illegal option or missing argument, respectively) and *OPTARG* is set to the option character that caused the problem. An error message is also printed to standard error if *optstring* does not begin with a colon.

When the end of the options is encountered, *getopts* exits with a non-zero exit status. Options end at the first (non-option argument) argument that does not start with a –, or when a **––** argument is encountered.

Option parsing can be reset by setting *OPTIND* to 1 (this is done automatically whenever the shell or a shell procedure is invoked).

Warning: Changing the value of the shell parameter *OPTIND* to a value other than 1, or parsing different sets of arguments without resetting *OPTIND* may lead to unexpected results.

Getopts is a built-in of the shell.

DIAGNOSTICS

Getopts returns a non-zero value when it encounters the end of the options.

EXAMPLE

This shell script echoes its arguments, one per line, unless the "–x" flag is given; options "–b" and "–x" without arguments and for "–o" with an argument.

```
CMD=$(basename $0)

USAGE="usage: $CMD [-bx] [-o argument] arg ..."

while getopts bo:x opt
do
        case "$opt" in
        'b') echo "-b specified"
             ;;
        'o') echo "-o $OPTARG specified"
             ;;
       'x') cat <<EndOfHelp
Name
Synopsis:
    $CMD [-bx] [-o argument] arg ..."

Options:
    -x   Print this help
EndOfHelp
             exit 1;;
        '?') echo $USAGE
             exit 1;;
        esac
done

shift $OPTIND-1

for arg in $@
do
       echo $arg
done
```

SEE ALSO

sh

NAME

grep, egrep, fgrep – file pattern searcher

SYNOPSIS

```
grep [-F|-E] [-c|-l|-q] [-hinosvx] -e pattern_list
     [-f pattern_file] ... [file ...]

grep [-F|-E] [-c|-l|-q] [-hinosvx] [-e pattern_list]
     -f pattern_file ... [file ...]

grep [-F|-E] [-c|-l|-q] [-hinosvx] pattern_list
     [-f pattern_file ...] [file ...]
```

DESCRIPTION

The *grep* utilities search the given input files selecting lines which match one or more patterns; the type of patterns is controlled by the options specified. By default, a pattern matches an input line if any regular expression (RE) in the pattern matches the input line without its trailing newline. A null RE matches every line. Each input line that matches at least one of the patterns is written to the standard output.

For simple patterns or *ex*- or *ed*-style regular expressions (basic regular expressions, described in the *INTERIX User's Guide*), use the *grep* utility. To use extended regular expressions, use the **–E** flag, or invoke the program as *egrep*. To search for a fixed string instead of a pattern, use the **–F** flag, or invoke the command as *fgrep*. A fixed string is a string of characters, each character is matched only by itself.

The following options are available:

–c Only a count of selected lines is written to standard output.

–E Use extended regular expressions.

–e *expression*
Specify a pattern used during the search of the input. Multiple **–e** options can be used to specify multiple patterns; an input line is selected if it matches any of the specified patterns.

–F Search for a fixed string rather than a regular expression. Cannot be used with **–E**.

–f *pattern_file*
The pattern is read from the file named by the pathname *pattern_file*. Trailing newlines in the pattern_file are ignored.

–h Suppress filenames in output. Normally, if more than one file is searched, *grep* prints the filename before printing the matched line. This option suppresses that behavior.

–i The case of letters is ignored in making comparisons — upper and lower case letters are considered identical.

–l List only the names of the files containing lines that match. Pathnames are listed once per file searched. If the standard input is searched, the pathname "stdin" is written. Useful for command lines like

```
vi 'grep -l -e "[Ee]xactly" *.txt'
```

–o Force filenames in output. Normally, if more than one file is searched, *grep* prints the filename before printing the matched line. This option forces the filename to be printed even if only one file is searched. For standard input, *grep* prints "(standard input):".

–q Quiet: write nothing to standard output and exit with 0 if a line is selected.

–s Silent mode. Nothing is printed (except error messages). This is useful for checking the error status.

–v Selected lines are those *not* matching the specified patterns.

–x "Exact" matching: the fixed string or the regular expression must match the entire line before it is considered a match. For example, the string "abc" matches a line that contains only "abc" but not a line that contains "abcd".

If no *file* arguments are specified, the standard input is used.

REGULAR EXPRESSIONS

Regular expressions are described in more detail on the man page *re_format*.

EXTENDED REGULAR EXPRESSIONS

The following characters are interpreted by *egrep*:

$ Align the match from the end of the line.

^ Align the match from the beginning of the line.

| Add another pattern (see example below).

? Match 1 or less sequential repetitions of the pattern.

+ Match 1 or more sequential repetitions of the pattern.

* Match 0 or more sequential repetitions of the pattern.

[] Match any single character or range of characters enclosed in the brackets.

\ Escape special characters which have meaning for extended regular expressions, the set of {$,.,^,[,],|,?,+,*,(,)}.

DIAGNOSTICS

The *grep* utility exits with one of the following values:

0 One or more lines were selected.

1 No lines were selected.

>1 An error occurred.

If the utility is invoked as *egrep*, then the **–E** flag is assumed. If the utility is invoked as *fgrep*, then the **–F** flag is assumed.

SEE ALSO

ed, *sed*

re_format

NAME

gzip, gunzip, uncompress, zcat – compress or expand files

SYNOPSIS

`gzip [ –acdfhlLnNrtvV19 ] [–S` *`suffix`* `] [`*`name ...`*`]`

`gunzip [ –acfhlLnNrtvV ] [ –S` *`suffix`* `] [ `*`name ...`*` ]`

`uncompress [–cfv] [`*`file ...`*`]`

`zcat [–fhLV] [`*`name ...`*`]`

DESCRIPTION

Gzip reduces the size of the named files using Lempel-Ziv coding (LZ77). Whenever possible, each file is replaced by one with the extension **.gz**, while keeping the same ownership modes, access and modification times. (The default extension is **.z** for Windows NT FAT.) If no files are specified, or if a file name is "-", the standard input is compressed to the standard output. *Gzip* will only attempt to compress regular files. In particular, it will ignore symbolic links.

If the compressed file name is too long for its file system, *gzip* truncates it. *Gzip* attempts to truncate only the parts of the file name longer than 3 characters. (A part is delimited by dots.) If the name consists of small parts only, the longest parts are truncated. For example, if file names are limited to 14 characters, **gzip.msdos.exe** is compressed to **gzi.msd.exe.gz**. Names are not truncated on systems which do not have a limit on file name length.

By default, *gzip* keeps the original file name and timestamp in the compressed file. These are used when decompressing the file with the **–N** option. This is useful when the compressed file name was truncated or when the time stamp was not preserved after a file transfer.

Compressed files can be restored to their original form using **gzip –d**, *gunzip*, *uncompress*, or *zcat*. If the original name saved in the compressed file is not suitable for its file system, a new name is constructed from the original one to make it legal.

gunzip takes a list of files on its command line and replaces each file whose name ends with **.gz** or **.Z** and which begins with the correct magic number with an uncompressed file without the original extension. *gunzip* also recognizes the special extensions **.tgz** and **.taz** as shorthands for **.tar.gz** and **.tar.Z** respectively. When compressing, *gzip* uses the **.tgz** extension if necessary instead of truncating a file with a **.tar** extension.

gunzip can currently decompress files created by *gzip*, *zip*, *compress*, *compress –H* or *pack*. The detection of the input format is automatic. When using the first two formats, *gunzip* checks a 32 bit CRC. For *pack*, *gunzip* checks the uncompressed length. The standard *compress* format was not designed to allow consistency checks. However *gunzip* is sometimes able to detect a bad **.Z** file. If you get an error when uncompressing a **.Z** file, do not assume that the **.Z** file is correct simply because the standard *uncompress* does not complain. This generally means that the standard *uncompress* does not check its input, and happily generates garbage output. The SCO `compress -H` format (lzh compression method) does not include a CRC but also allows some consistency checks.

Files created by *zip* can be uncompressed by *gzip* only if they have a single member compressed with the 'deflation' method. This feature is only intended to help conversion of tar.zip files to the tar.gz format. To extract zip files with several members,

use *unzip* instead of *gunzip*.

uncompress is identical to *gunzip* except that the

zcat is identical to *gunzip* **–c**. (On some systems, *zcat* may be installed as *gzcat* to preserve the original link to *compress.) zcat* uncompresses either a list of files on the command line or its standard input and writes the uncompressed data on standard output. *zcat* will uncompress files that have the correct magic number whether they have a **.gz** suffix or not.

Gzip uses the Lempel-Ziv algorithm used in *zip* and PKZIP. The amount of compression obtained depends on the size of the input and the distribution of common substrings. Typically, text such as source code or English is reduced by 60–70%. Compression is generally much better than that achieved by LZW (as used in *compress*), Huffman coding (as used in *pack*), or adaptive Huffman coding (*compact*).

Compression is always performed, even if the compressed file is slightly larger than the original. The worst case expansion is a few bytes for the *gzip* file header, plus 5 bytes every 32K block, or an expansion ratio of 0.015% for large files. Note that the actual number of used disk blocks almost never increases. *gzip* preserves the mode, ownership and timestamps of files when compressing or decompressing.

OPTIONS

Gzip takes the following options. The long form of the option letter is listed in parentheses; the long forms are preceded by *two* dashes (––).

–c (––stdout ––to-stdout)
Write output on standard output; keep original files unchanged. If there are several input files, the output consists of a sequence of independently compressed members. To obtain better compression, concatenate all input files before compressing them.

–d (––decompress ––uncompress)
Decompress.

–f (––force)
Force compression or decompression even if the file has multiple links or the corresponding file already exists, or if the compressed data is read from or written to a terminal. If the input data is not in a format recognized by *gzip*, and if the option **––stdout** is also given, copy the input data without change to the standard output: let *zcat* behave as *cat*. If **–f** is not given, and when not running in the background, *gzip* prompts to verify whether an existing file should be overwritten.

–h (––help)
Display a help screen and quit.

–l (––list)
For each compressed file, list the following fields:

`compressed size`: size of the compressed file

`uncompressed size`: size of the uncompressed file

`ratio`: compression ratio (0.0% if unknown)

`uncompressed_name`: name of the uncompressed file
The uncompressed size is given as –1 for files not in gzip format, such as compressed **.Z** files. To get the uncompressed size for such a file, you can use:

```
$ zcat file.Z | wc -c
```

In combination with the **––verbose** option, the following fields are also displayed:

`method`: compression method

`crc`: the 32-bit CRC of the uncompressed data

`date & time`: time stamp for the uncompressed file
The compression methods currently supported are deflate, compress, lzh (SCO **compress –H**) and pack. The crc is given as ffffffff for a file not in gzip format.

With **––name**, the uncompressed name, date and time are those stored within the compress file if present.

With **––verbose**, the size totals and compression ratio for all files is also displayed, unless some sizes are unknown. With **––quiet**, the title and totals lines are not displayed.

–L (––license)
Display the *gzip* license and quit.

–n (––no-name)
When compressing, do not save the original file name and time stamp by default. (The original name is always saved if the name had to be truncated.) When decompressing, do not restore the original file name if present (remove only the *gzip* suffix from the compressed file name) and do not restore the original time stamp if present (copy it from the compressed file). This option is the default when decompressing.

–N (––name)
When compressing, always save the original file name and time stamp; this is the default. When decompressing, restore the original file name and time stamp if present. This option is useful on systems which have a limit on file name length or when the time stamp has been lost after a file transfer.

–q (––quiet)
Suppress all warnings.

–r (––recursive)
Travel the directory structure recursively. If any of the file names specified on the command line are directories, *gzip* will descend into the directory and compress all the files it finds there (or decompress them in the case of *gunzip*).

–S *.suf* **(—suffix** *.suf***)**
Use suffix *.suf* instead of **.gz**. Any suffix can be given, but suffixes other than **.z** and **.gz** should be avoided to avoid confusion when files are transferred to other systems. A null suffix forces *gunzip* to try decompression on all given files regardless of suffix, as in:

```
gunzip -S "" *
```

Previous versions of gzip used the **.z** suffix. This was changed to avoid a conflict with *pack*.

–t (––test)
Test. Check the compressed file integrity.

–v (––verbose)
Verbose. Display the name and percentage reduction for each file compressed or decompressed.

–V (––version)
Version. Display the version number and compilation options then quit.

–# (––fast ––best)
Regulate the speed of compression using the specified digit #, where **–1** or **—fast** indicates the fastest compression method (less compression) and **–9** or **—best** indicates the slowest compression method (best compression). The default compression level is **–6** (that is, biased towards high compression at expense of speed).

ADVANCED USAGE

Multiple compressed files can be concatenated. In this case, *gunzip* will extract all members at once. For example:

```
gzip –c file1  > foo.gz
gzip –c file2 >> foo.gz
```

Then

```
gunzip –c foo
```

is equivalent to

```
cat file1 file2
```

In case of damage to one member of a **.gz** file, other members can still be recovered (if the damaged member is removed). However, you can get better compression by compressing all members at once:

```
cat file1 file2 | gzip > foo.gz
```

compresses better than

```
gzip –c file1 file2 > foo.gz
```

If you want to recompress concatenated files to get better compression, do:

```
gzip –cd old.gz | gzip > new.gz
```

If a compressed file consists of several members, the uncompressed size and CRC reported by the **––list** option applies to the last member only. If you need the uncompressed size for all members, you can use:

```
gzip –cd file.gz | wc –c
```

If you wish to create a single archive file with multiple members so that members can later be extracted independently, use an archiver such as *tar* or *zip*. GNU *tar* supports the **–z** option to invoke *gzip* transparently. *gzip* is designed as a complement to *tar*, not as a replacement.

ENVIRONMENT VARIABLES

The environment variable *GZIP* can hold a set of default options for *gzip*. These options are interpreted first and can be overwritten by explicit command line parameters. For example:

for sh: `GZIP="-8v --name"; export GZIP`
for csh: `setenv GZIP "-8v --name"`

DIAGNOSTICS

Exit status is normally 0; if an error occurs, exit status is 1. If a warning occurs, exit status is 2.

Usage: gzip [-cdfhlLnNrtvV19] [-S suffix] [file ...]
Invalid options were specified on the command line.

file: not in gzip format
The file specified to *gunzip* has not been compressed.

file: Corrupt input. Use zcat to recover some data.
The compressed file has been damaged. The data up to the point of failure can be recovered using

`zcat file > recover`

file: compressed with *xx* bits, can only handle *yy* bits
File was compressed (using LZW) by a program that could deal with more *bits* than the decompress code on this machine. Recompress the file with gzip, which compresses better and uses less memory.

file: already has .gz suffix -- no change
The file is assumed to be already compressed. Rename the file and try again.

file already exists; do you wish to overwrite (y or n)?
Respond "y" if you want the output file to be replaced; "n" if not.

gunzip: corrupt input
A SIGSEGV violation was detected which usually means that the input file has been corrupted.

xx.x%
Percentage of the input saved by compression. (Relevant only for **–v** and **–l**.)

-- not a regular file or directory: ignored
When the input file is not a regular file or directory, (e.g. a symbolic link, socket, FIFO, device file), it is left unaltered.

-- has *xx* other links: unchanged
The input file has links; it is left unchanged. See *ln* for more information. Use the **–f** flag to force compression of multiply-linked files.

CAVEATS

When writing compressed data to a tape, it is generally necessary to pad the output with zeroes up to a block boundary. When the data is read and the whole block is passed to *gunzip* for decompression, *gunzip* detects that there is extra trailing garbage after the compressed data and emits a warning by default. You have to use the **––quiet** option to suppress the warning. This option can be set in the *GZIP* environment variable as in:

for sh: `GZIP="-q" tar -xfz --block-compress /dev/rst0`
for csh: `(setenv GZIP -q; tar -xfz --block-compr /dev/rst0)`

In the above example, *gzip* is invoked implicitly by the **–z** option of GNU *tar*. Make sure that the same block size (the **–b** option of *tar*) is used for reading and writing compressed data on tapes. (This example assumes you are using the GNU version of *tar*.)

BUGS

The **––list** option reports incorrect sizes if they exceed 2 gigabytes. The **––list** option reports sizes as -1 and crc as ffffffff if the compressed file is on a non-seekable medium.

In some rare cases, the **––best** option gives worse compression than the default compression level (–6). On some highly redundant files, traditional *compress* compresses better than *gzip*; this is not true on INTERIX, where *compress* is a link to *gzip*.

SEE ALSO

zip, *unzip*

NAME

head – display first lines of a file

SYNOPSIS

`head [-n count] [ file ...]`

DESCRIPTION

This filter displays the first *count* lines of each of the specified files, or of the standard input if no files are specified. If *count* is omitted it defaults to 10.

If more than a single file is specified, each file is preceded by a header consisting of the string ‘‘==> *XXX* <==’’ where *XXX* is the name of the file.

DIAGNOSTICS

The *head* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

tail

NAME

hexdump – ascii, decimal, hexadecimal, octal dump

SYNOPSIS

```
hexdump [-bcdovx] [-e format_string] [-f format_file]
        [-n length] -words [-s skip] file ...
```

DESCRIPTION

The *hexdump* utility is a filter which displays the specified files, or the standard input, if no files are specified, in a user specified format.

The options are as follows:

–b *One-byte octal display*. Display the input offset in hexadecimal, followed by sixteen space-separated, three column, zero-filled, bytes of input data, in octal, per line.

–c *One-byte character display*. Display the input offset in hexadecimal, followed by sixteen space-separated, three column, space-filled, characters of input data per line.

–d *Two-byte decimal display*. Display the input offset in hexadecimal, followed by eight space-separated, five column, zero-filled, two-byte units of input data, in unsigned decimal, per line.

–e *format_string*
Specify a *fprintf*( )-style format string to be used for displaying data.

–f *format_file*
Specify a file that contains one or more newline separated format strings. Empty lines and lines whose first non-blank character is a hash mark (#) are ignored.

–n *length*
Interpret only *length* bytes of input.

–o *Two-byte octal display*. Display the input offset in hexadecimal, followed by eight space-separated, six column, zero-filled, two byte quantities of input data, in octal, per line.

–s *offset* Skip *offset* bytes from the beginning of the input. By default, *offset* is interpreted as a decimal number. With a leading **0x** or **0X**, *offset* is interpreted as a hexadecimal number, otherwise, with a leading **0**, *offset* is interpreted as an octal number. Appending the character **b**, **k**, or **m** to *offset* causes it to be interpreted as a multiple of 512, 1024, or 1048576, respectively.

–v The **–v** option causes hexdump to display all input data. Without the **–v** option, any number of groups of output lines, which would be identical to the immediately preceding group of output lines (except for the input offsets), are replaced with a line comprised of a single asterisk.

–x *Two-byte hexadecimal display*. Display the input offset in hexadecimal, followed by eight, space separated, four column, zero-filled, two-byte quantities of input data, in hexadecimal, per line.

For each input file, *hexdump* sequentially copies the input to standard output, transforming the data according to the format strings specified by the **–e** and **–f** options,

in the order that they were specified. A format string contains any number of format units, separated by whitespace. A format unit contains up to three items: an iteration count, a byte count, and a format.

The iteration count is an optional positive integer, which defaults to one. Each format is applied iteration count times.

The byte count is an optional positive integer. If specified it defines the number of bytes to be interpreted by each iteration of the format.

If an iteration count and / or a byte count is specified, a single slash must be placed after the iteration count and / or before the byte count to disambiguate them. Any whitespace before or after the slash is ignored.

The format is required and must be surrounded by double quote (" ") marks. It is interpreted as a *fprintf*( )-style format string, with the following exceptions:

- An asterisk (*) may not be used as a field width or precision.
- A byte count or field precision *is* required for each "s" conversion character (unlike the *fprintf* default which prints the entire string if the precision is unspecified).
- The conversion characters "h", "l", "n", "p" and "q" are not supported.
- The single character escape sequences described in the ISO / ANSI C standard are supported:

NUL	\0
alert character	\a
backspace	\b
form-feed	\f
newline	\n
carriage return	\r
tab	\t
vertical tab	\v

Hexdump also supports the the following additional conversion strings:

_a[**dox**] Display the input offset, cumulative across input files, of the next byte to be displayed. The appended characters **d**, **o**, and **x** specify the display base as decimal, octal or hexadecimal respectively.

_A[**dox**] **Identical to the _a conversion string except that it is only performed once, when all of the input data has been processed.**

_c **Output characters in the default character set. Nonprinting characters are displayed in three character, zero-padded octal, except for those representable by standard escape notation (see above), which are displayed as two character strings.**

_p Output characters in the default character set. Nonprinting characters are displayed as a single ".".

_u Output US ASCII characters, with the exception that control characters are displayed using the following, lower-case, names. Characters greater than 0xff, hexadecimal, are displayed as hexadecimal strings.

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq
006 ack	007 bel	008 bs	009 ht	00A lf	00B vt
00C ff	00D cr	00E so	00F si	010 dle	011 dc1
012 dc2	013 dc3	014 dc4	015 nak	016 syn	017 etb
018 can	019 em	01A sub	01B esc	01C fs	01D gs
01E rs	01F us	0FF del			

The default and supported byte counts for the conversion characters are as follows:

%_c, %_p, %_u, %c	One byte counts only.
%d, %i, %o, %u, %X, %x	Four byte default, one, two and four byte counts supported.
%E, %e, %f, %G, %g	Eight byte default, four byte counts supported.

The amount of data interpreted by each format string is the sum of the data required by each format unit, which is the iteration count times the byte count, or the iteration count times the number of bytes required by the format if the byte count is not specified.

The input is manipulated in “blocks”, where a block is defined as the largest amount of data specified by any format string. Format strings interpreting less than an input block’s worth of data, whose last format unit both interprets some number of bytes and does not have a specified iteration count, have the iteration count incremented until the entire input block has been processed or there is not enough data remaining in the block to satisfy the format string.

If, either as a result of user specification or hexdump modifying the iteration count as described above, an iteration count is greater than one, no trailing whitespace characters are output during the last iteration.

It is an error to specify a byte count as well as multiple conversion characters or strings unless all but one of the conversion characters or strings is **_a** or **_A**.

If, as a result of the specification of the **–n** option or end-of-file being reached, input data only partially satisfies a format string, the input block is zero-padded sufficiently to display all available data (i.e. any format units overlapping the end of data will display some number of the zero bytes).

Further output by such format strings is replaced by an equivalent number of spaces. An equivalent number of spaces is defined as the number of spaces output by an **s** conversion character with the same field width and precision as the original conversion character or conversion string but with any “+”, “ ”, “#” conversion flag characters removed, and referencing a NULL string.

If no format strings are specified, the default display is equivalent to specifying the **–x** option.

DIAGNOSTICS

hexdump exits 0 on success and >0 if an error occurred.

SEE ALSO

od

NAME

hostname – print name of current host system

SYNOPSIS

`hostname [-ais]`

DESCRIPTION

Hostname prints the name of the current host. Unlike traditional implementations, this version cannot be used to set the hostname. See Windows NT documentation for instructions on setting the hostname.

Options:

–a Print all addresses for your machine.

–i Print curent IP address for your machine.

–s Trim off any domain information from the printed name.

SEE ALSO

gethostname()

NAME

ico – animate an icosahedron or other polyhedron

SYNOPSIS

```
ico [-display display] [-geometry geometry] [-r] [-d pattern]
    [-i] [-dbl] [-faces] [-noedges]
    [-sleep n] [-obj object] [-objhelp] [-colors color-list]
```

DESCRIPTION

Ico displays a wire-frame rotating polyhedron, with hidden lines removed, or a solid-fill polyhedron with hidden faces removed. There are a number of different polyhedra available; adding a new polyhedron to the program is quite simple.

OPTIONS

–r Display on the root window instead of creating a new window.

–d *pattern*
Specify a bit pattern for drawing dashed lines for wire frames.

–i Use inverted colors for wire frames.

–dbl Use double buffering on the display. This works for either wire frame or solid fill drawings. For solid fill drawings, using this switch results in substantially smoother movement. Note that this requires twice as many bit planes as without double buffering. Since some colors are typically allocated by other programs, most eight-bit-plane displays will probably be limited to eight colors when using double buffering.

–faces Draw filled faces instead of wire frames.

–noedges
Don't draw the wire frames. Typically used only when -faces is used.

–sleep *n*
Sleep n seconds between each move of the object.

–obj *object*
Specify what object to draw. If no object is specified, an icosahedron is drawn.

–objhelp
Print out a list of the available objects, along with information about each object.

–colors *color*
Specify what colors should be used to draw the filled faces of the object. If less colors than faces are given, the colors are reused.

ADDING POLYHEDRA

If you have the source to ico, it is very easy to add more polyhedra. Each polyhedron is defined in an include file by the name of objXXX.h, where XXX is something related to the name of the polyhedron. The format of the include file is defined in the file polyinfo.h. Look at the file objcube.h to see what the exact format of an objXXX.h file should be, then create your objXXX.h file in that format.

After making the new objXXX.h file (or copying in a new one from elsewhere), simply do a 'make depend'. This will recreate the file allobjs.h, which lists all of the objXXX.h

files. Doing a 'make' after this will rebuild ico with the new object information.

SEE ALSO

X

BUGS

Pyramids and tetrahedrons with filled faces do not display correctly.

A separate color cell is allocated for each name in the -colors list, even when the same name may be specified twice.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

NAME

id – return user identity

SYNOPSIS

```
id [user]
id -G [-n] [user]
id -g [-nr] [user]
id -u [-nr] [user]
```

DESCRIPTION

The *id* utility displays the user and group names and numeric IDs of the calling process to the standard output. If the real and effective IDs are different, both are displayed, otherwise only the real ID is displayed.

If you specify a *user* (by login name or user ID) *id* displays the user and group IDs of that user. In this case, the real and effective IDs are assumed to be the same.

The options are as follows:

–G Display the different group IDs (effective, real and supplementary) as white-space separated numbers, in no particular order.

–g Display the effective group ID as a number.

–n Display the name of the user or group ID for the **–G**, **–g** and **–u** options instead of the number. If any of the ID numbers cannot be mapped into names, the number will be displayed as usual.

–r Display the real ID for the **–g** and **–u** options instead of the effective ID.

–u Display the effective user ID as a number.

The traditional BSD command *whoami* is the same as

```
id -un
```

DIAGNOSTICS

The *id* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

logname

NAME

ident – identify files

SYNOPSIS

`ident [ –q ] [ file . . . ]`

DESCRIPTION

ident searches for all occurrences of the pattern **$***keyword***:**. . .**$** in the named files or, if no file is specified, the standard input.

These patterns are normally inserted automatically by the RCS command *co*, but can also be inserted manually. The option **–q** suppresses the warning given if there are no patterns in a file.

Ident works on text files as well as object files and dumps. For example, if the C program in **f.c** contains

```
char rcsid[] = "$Id: f.c,v 1.5 1997/03/13 22:07:10 johnmc Exp $";
```

and **f.c** is compiled into **f.o**, then the command

```
$ ident  f.c  f.o
f.c:
     $Id: f.c,v 1.5 1997/03/13 22:07:10 johnmc Exp $
f.o:
     $Id: f.c,v 1.5 1997/03/13 22:07:10 johnmc Exp $
```

DIAGNOSTICS

Ident exits with status 0 on success and 1 on failure.

IDENTIFICATION

Author: Walter F. Tichy.

Copyright © 1982, 1988, 1989 by Walter F. Tichy.

Copyright © 1990 by Paul Eggert.

SEE ALSO

ci, *co*, *rcs*, *rcsdiff*, *rlog*

Walter F. Tichy, RCS—A System for Version Control, *Software—Practice & Experience* **15**, 7 (July 1985), 637-654.

NAME

imake – C preprocessor interface to the make utility

SYNOPSIS

```
imake [-Ddefine] [-Idir] [-Ttemplate]
      [-f filename] [-s filename] [-e ] [ -v]
```

DESCRIPTION

Imake is used to generate **Makefile**s from a template, a set of *cpp* macro functions, and a per-directory input file called an **Imakefile**. This allows machine dependencies (such has compiler options, alternate command names, and special *ake* rules) to be kept separate from the descriptions of the various items to be built.

OPTIONS

The following command line options may be passed to *imake*:

–D*define*

This option is passed directly to *cpp*. It is typically used to set directory-specific variables. For example, the X Window System uses this flag to set *TOPDIR* to the name of the directory containing the top of the core distribution and *CURDIR* to the name of the current directory, relative to the top.

–I*directory*

This option is passed directly to *cpp*. It is typically used to indicate the directory in which the *imake* template and configuration files may be found.

–T*template*

This option specifies the name of the master template file (which is usually located in the directory specified with **–I**) used by *cpp*. The default is **Imake.tmpl**.

–f*filename*

This option specifies the name of the per-directory input file. The default is **Imakefile**.

–s*filename*

This option specifies the name of the *make* description file to be generated but *make* should not be invoked. If the *filename* is a dash (–), the output is written to *stdout*. The default is to generate, but not execute, a *Makefile*.

–e This option indicates the *imake* should execute the generated **Makefile**. The default is to leave this to the user.

–v This option indicates that *imake* should print the *cpp* command line that it is using to generate the **Makefile**.

HOW IT WORKS

Imake invokes *cpp* with any **–I** or **–D** flags passed on the command line and passes it the following 3 lines:

```
#define IMAKE_TEMPLATE "Imake.tmpl"
#define INCLUDE_IMAKEFILE "Imakefile"
#include IMAKE_TEMPLATE
```

where **Imake.tmpl** and **Imakefile** may be overridden by the **–T** and **–f** command

options, respectively.

The IMAKE_TEMPLATE typically reads in a file containing machine-dependent parameters (specified as *cpp* symbols), a site-specific parameters file, a file defining variables, a file containing *cpp* macro functions for generating *make* rules, and finally the **Imakefile** (specified by INCLUDE_IMAKEFILE) in the current directory. The **Imakefile** uses the macro functions to indicate what targets should be built; *imake* takes care of generating the appropriate rules.

Imake configuration files contain two types of variables, imake variables and make variables. The imake variables are interpreted by cpp when *imake* is run. By convention they are mixed case. The make variables are written into the **Makefile** for later interpretation by *make*. By convention make variables are upper case.

The rules file (usually named **Imake.rules** in the configuration directory) contains a variety of *cpp* macro functions that are configured according to the current platform. *Imake* replaces any occurrences of the string "@@" with a newline to allow macros that generate more than one line of *make* rules. For example, the macro

```
#define       program_target(program, objlist)  @@\
program:      objlist              @@\
      $(CC)  -o  $@  objlist  $(LDFLAGS)
```

when called with *program_target(foo, foo1.o foo2.o)* will expand to

```
foo:   foo1.o  foo2.o
       $(CC)  -o  $@  foo1.o  foo2.o  $(LDFLAGS)
```

On systems whose *cpp* reduces multiple tabs and spaces to a single space, *imake* attempts to put back any necessary tabs (*make* is very picky about the difference between tabs and spaces). For this reason, colons (:) in command lines must be preceded by a backslash (\).

USE WITH THE X WINDOW SYSTEM

The X Window System uses *imake* extensively, for both full builds within the source tree and external software. As mentioned above, two special variables, *TOPDIR* and *CURDIR*, are set to make referencing files using relative path names easier. For example, the following command is generated automatically to build the **Makefile** in the directory **lib/X/** (relative to the top of the sources):

```
%  ../../../config/imake  -I../../../config  \
       -DTOPDIR=../../.   -DCURDIR=./lib/X
```

When building X programs outside the source tree, a special symbol *UseInstalled* is defined and *TOPDIR* and *CURDIR* are omitted. If the configuration files have been properly installed, the script *xmkmf*(1) may be used.

INPUT FILES

Here is a summary of the files read by *imake* as used by X. The indentation shows what files include what other files.

```
Imake.tmpl          generic variables
    site.def        site-specific, BeforeVendorCF defined
    *.cf            machine-specific
        *Lib.rules  shared library rules
    site.def        site-specific, AfterVendorCF defined
    Project.tmpl    X-specific variables
```

```
            *Lib.tmpl   shared library variables
        Imake.rules     rules
    Imakefile
        Library.tmpl    library rules
        Server.tmpl     server rules
```

Note that **site.def** gets included twice, once before the ***.cf** file and once after. Although most site customizations should be specified after the ***.cf** file, some, such as the choice of compiler, need to be specified before, because other variable settings may depend on them.

The first time **site.def** is included, the variable BeforeVendorCF is defined, and the second time, the variable AfterVendorCF is defined. All code in **site.def** should be inside an #ifdef for one of these symbols.

FILES

Imake uses the following files:

$OPENNT_ROOT/usr/tmp/tmp-imake.*nnnnnn*
Temporary input file for *cpp*.

$OPENNT_ROOT/usr/tmp/tmp-make.*nnnnnn*
Temporary input file for *make*.

$OPENNT_ROOT/usr/contrib/cpp
Default C preprocessor.

$OPENNT_ROOT/usr/X11R5/lib/X11/config
Default configuration and template files for INTERIX systems.

ENVIRONMENT VARIABLES

The following environment variables may be set, however their use is not recommended as they introduce dependencies that are not readily apparent when *imake* is run:

IMAKEINCLUDE
If defined, this should be a valid include argument for the C preprocessor. E.g., "–I//E/OpenNT/usr/include/local". Actually, any valid *cpp* argument will work here.

IMAKECPP
If defined, this should be a valid path to a preprocessor program. E.g. "//C/OpenNT/usr/local/cpp". By default, *imake* will use **$OPENNT_ROOT/usr/contrib/***machine*/*version*, where *machine* is either **alpha-pc-opennt** (for DEC Alpha machines) or **intel-pc-opennt** (for X86 machines), and *version* is the version of **gcc**.

IMAKEMAKE
If defined, this should be a valid path to a make program, such as "//C/OpenNT/bin/make". By default, *imake* will use whatever *make* program is found using *execvp*( ). This variable is only used if the **–e** option is specified.

AUTHOR

Todd Brunhoff, Tektronix and MIT Project Athena; Jim Fulton, MIT X Consortium

SEE ALSO

make, *xmkmf*

S. I. Feldman, *Make — A Program for Maintaining Computer Programs*

NAME

inetd – internet "super-server"

SYNOPSIS

`inetd [-ds] [-R rate] [configuration file]`

DESCRIPTION

The *inetd* program listens for connections on certain internet sockets. When a connection is found on one of its sockets, it decides what service the socket corresponds to, and invokes a program to service the request. The server program is invoked with the service socket as its standard input, output and error descriptors. After the program is finished, *inetd* continues to listen on the socket (except in some cases which will be described below). Essentially, *inetd* allows running one daemon to invoke several others, reducing load on the system.

Traditionally, *inetd* is run at boot time by **/etc/rc**; however, on INTERIX, it is usually installed as a service.

The options available for *inetd:*

–d Turn on debugging.

–s Run as a service (do not fork upon startup). This option is *required* to run *inetd* as a Windows NT service.

–R *rate* Specify the maximum number of times a service can be invoked in one minute; the default is 1000.

Upon execution, *inetd* reads its configuration information from a configuration file which, by default, is **$OPENNT_ROOT/etc/inetd.conf**. There must be an entry for each field of the configuration file, with entries for each field separated by a tab or a space. Comments are denoted by a ''#'' at the beginning of a line. There must be an entry for each field. The fields of the configuration file are as follows:

name sock_type protocol wait user serv_prog serv_prog_args

There are two types of services that *inetd* can start: standard and TCPMUX. A standard service has a well-known port assigned to it; it may be a service that implements an official Internet standard or is a BSD-specific service. As described in RFC 1078, TCPMUX services are nonstandard services that do not have a well-known port assigned to them. They are invoked from *inetd* when a program connects to the "tcpmux" well-known port and specifies the service name. This feature is useful for adding locally-developed servers.

name The *service-name* entry is the name of a valid service in the file **%WINDIR%/system32/drivers/etc/services**, where **%WINDIR%** is the Windows NT installation directory. For "internal" services (discussed below), the service name *must* be the official name of the service (that is, the first field in the entry in **%WINDIR%/system32/drivers/etc/services**). For TCPMUX services, the value of the *service-name* field consists of the string "tcpmux" followed by a slash and the locally-chosen service name. The service names listed in **%WINDIR%/system32/drivers/etc/services** and the name "help" are reserved. Try to choose unique names for your TCPMUX services by prefixing them with your organization's name and suffixing them with a version number.

sock_type

The *socket-type* should be one of “stream”, “dgram”, “raw”, “rdm”, or “seqpacket”, depending on whether the socket is a stream, datagram, raw, reliably delivered message, or sequenced packet socket. TCPMUX services must use “stream.”

protocol The *protocol* must be a valid protocol as given in **%WINDIR%/system32/drivers/etc/protocols.** Examples might be “tcp” or “udp”. TCPMUX services must use “tcp”.

wait The *wait/nowait* entry specifies whether the server that is invoked by *inetd* will take over the socket associated with the service access point, and thus whether *inetd* should wait for the server to exit before listening for new service requests. Datagram servers must use “wait”, as they are always invoked with the original datagram socket bound to the specified service address. These servers must read at least one datagram from the socket before exiting. If a datagram server connects to its peer, freeing the socket so *inetd* can received further messages on the socket, it is said to be a “multi-threaded” server; it should read one datagram from the socket and create a new socket connected to the peer. It should fork, and the parent should then exit to allow *inetd* to check for new service requests to spawn new servers. Datagram servers which process all incoming datagrams on a socket and eventually time out are said to be “single-threaded.” Both *biff* and *ntalkd* are examples of the latter type of datagram server. *Tftpd* is an example of a multi-threaded datagram server.

Servers using stream sockets generally are multi-threaded and use the “nowait” entry. Connection requests for these services are accepted by *inetd*, and the server is given only the newly-accepted socket connected to a client of the service. Most stream-based services operate in this manner. Stream-based servers that use “wait” are started with the listening service socket, and must accept at least one connection request before exiting. Such a server would normally accept and process incoming connection requests until a timeout. TCPMUX services must use “nowait”.

user

Historically, the *user* entry should contain the user name of the user as whom the server should run. This allows for servers to be given less permission than root. On INTERIX, this field is always set to the string “NULL” and is ignored. The *inetd* should always run as a low privilege user (for example, a global user named “inetd” with only Local Logon Privilege).

serv_prog

The *server-program* entry should contain the pathname of the program which is to be executed by *inetd* when a request is found on its socket. This pathname is relative to the installation directory for INTERIX (**$OPENNT_ROOT**). Do **not** give the path represented by **$OPENNT_ROOT**. For example, if the program is **$OPENNT_ROOT/bin/ftpd**, the pathname should read **/bin/ftpd**. If *inetd* provides this service internally, this entry should be “internal”.

serv_prog_args

The *server program arguments* should be just as arguments normally are, starting with argv[0], which is the name of the program. If the service is provided internally, the word “internal” should take the place of this entry.

The *inetd* program provides several “trivial” services internally by use of routines within itself. These services are “echo”, “discard”, “chargen” (character generator), “daytime” (human readable time), and “time” (machine readable time, in the form of the number of

seconds since midnight, January 1, 1900). All of these services are tcp based. For details of these services, consult the appropriate RFC from the Network Information Center.

The *inetd* program rereads its configuration file when it receives a hangup signal, SIGHUP. Services may be added, deleted or modified when the configuration file is reread.

TCPMUX

RFC 1078 describes the TCPMUX protocol: ''A TCP client connects to a foreign host on TCP port 1. It sends the service name followed by a carriage-return line-feed <CRLF>. The service name is never case sensitive. The server replies with a single character indicating positive (+) or negative (–) acknowledgment, immediately followed by an optional message of explanation, terminated with a <CRLF>. If the reply was positive, the selected protocol begins; otherwise the connection is closed.'' The program is passed the TCP connection as file descriptors 0 and 1.

If the TCPMUX service name begins with a ''+'', *inetd* returns the positive reply for the program. This allows you to invoke programs that use stdin/stdout without putting any special server code in them.

The special service name "help" causes *inetd* to list TCPMUX services in **inetd.conf**.

EXAMPLES

Here are several example service entries for the various types of services:

```
ftp           stream  tcp   nowait root  /usr/libexec/ftpd       ftpd -l
ntalk         dgram   udp   wait   root  /usr/libexec/ntalkd     ntalkd
tcpmux/+date  stream  tcp   nowait guest /bin/date                date
tcpmux/phonebook stream tcp nowait guest /usr/local/bin/phonebook phonebook
```

ERROR MESSAGES

The *inetd* server logs error messages using *syslog*( ); you may want to start *syslogd*. Important error messages and their explanations are:

service/*protocol* server failing (looping), service terminated.

The number of requests for the specified service in the past minute exceeded the limit. The limit exists to prevent a broken program or a malicious user from swamping the system. This message may occur for several reasons:

1. there are lots of hosts requesting the service within a short time period,
2. a 'broken' client program is requesting the service too frequently,
3. a malicious user is running a program to invoke the service in a 'denial of service' attack, or
4. the invoked service program has an error that causes clients to retry quickly. Use the **–R** option, as described above, to change the rate limit. Once the limit is reached, the service will be reenabled automatically in 10 minutes.

service/*protocol*: No such user '*user*', service ignored

service/*protocol*: getpwnam: *user*: No such user

No entry for *user* exists in the password database. The first message occurs when *inetd* (re)reads the configuration file. The second message occurs when the service is invoked.

service: can't set uid *number*

service: can't set gid *number*

The user or group ID for the entry's *user* is invalid.

NOTES

The *telnetd* daemon must be started with the **–i** option to run from *inetd*.

You can disable *all* daemons spawned by *inetd* by using the command

```
service stop inetd
```

This is a useful security measure.

SEE ALSO

fingerd, *ftpd*, *ntalkd*, *rexecd*, *rlogind*, *rshd*, *service*, *telnetd*

accept(), *bind*(), *connect*(), *listen*(), *socket*()

NAME

infocmp – compare or print out terminfo descriptions

SYNOPSIS

```
infocmp [-cdeFinpILCuV1] [–A directory] [–B directory]
        [–s d|i|l|c] [-v n] [–w width]
        [termname...]
```

DESCRIPTION

infocmp can be used to compare a binary **terminfo** entry with other terminfo entries, rewrite a **terminfo** description to take advantage of the **use=** terminfo field, or print out a **terminfo** description from the binary file (**term**) in a variety of formats. In all cases, the boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termname*s are specified, the **–-I** option will be assumed. If more than one *termname* is specified, the **–d** option will be assumed.

The options are summarized here and explained in more detail later.

–1 Print fields one to a line; the default is to print several to a line to a maximum line length of 60 characters.

–A *directory*
Treats *directory* as the location of the compiled *terminfo* database for the first *termname* given. By default, the location of the databse is taken from the enviornment variable *TERMINFO*.

–B *directory*
Treats *directory* as the location of the compiled *terminfo* database for all *termname*s given after the first. By default, the location of the databse is taken from the enviornment variable *TERMINFO*.

–C Write a source listing for each terminal named, using the *termcap* names.

–c Produce a list of each capability that is common between two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **–u** option is worth using.

–d Produce a list of each capability that is different between two entries. This option is useful to show the difference between two entries, created by different people, for the same or similar terminals.

–e Dump capabilities of the given terminal as a C initializer for a **TERMTYPE** structure.

–F *file1 file2*
Compare the terminfo files *file1* and *file2*.

–I Write a source listing for each terminal named, using the *terminfo* names.

–i Analyze the initialization and reset strings in the entry.

–L Write a source listing for each terminal named, using the long C variable names.

–n List each capability found in neither of the two entries.

–p Ignore padding specifications when comparing strings.

–R *subset* Restrict output to a given subset. Subsets are based on platform; available values for *subset* are **BSD**, **SVr1**, **Ultrix**, **HP**, and **fAIX**.

–r Modifies **–C**; writes out all capabilities in the *termcap* form.

–s *type* Sort fields within each type according to the *type* argument:

- **c** Sort by the *termcap* name .
- **d** Do not sort; leave in the order they occur in the *terminfo* database.
- **i** Sort by *terminfo* name.
- **l** Sort by long C variable name.

–u Produce a source description for the first *termname* given which is relative to descriptions for all other *termname*s given. That is, *infocmp* writes a source description for the first *termname* which attempts to **use=** the other *termname*s.

–V Print version of *infocmp* to standard error and exit.

–v *n* Print out tracing infomration. Higher values of *n* produce more verbosity.

–w *width* Change the output to *width* characters.

Comparison Options [-d] [-c] [-n]

infocmp compares the **terminfo** description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability:

F for boolean variables,
–1 for integer variables,
NULL
for string variables.

–c Produce a list of each capability that is common between two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **–u** option is worth using.

–d Produce a list of each capability that is different between two entries. This option is useful to show the difference between two entries, created by different people, for the same or similar terminals.

–n Produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable **TERM** will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of a description.

Source Listing Options [-C] [-I] [-L] [-r]

The **–C**, **–I**, and **–L** options will produce a source listing for each terminal named. The **–r** modifies **–C**:

–C Use the **termcap** names.

–I Use the **terminfo** names.

–L Use the long C variable name listed in **<term.h>**.

–r When using **–C**, put out all capabilities in **termcap** form.

If no *termname*s are given, the environment variable *TERM* will be used for the terminal name.

The source produced by the **–C** option may be used directly as a **termcap** entry, but not all parameterized strings can be changed to the **termcap** format. *infocmp* will attempt to convert most of the parameterized information, and anything not converted will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where **termcap** expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All **termcap** variables no longer supported by **terminfo**, but which are derivable from other **terminfo** variables, will be output. Not all **terminfo** capabilities will be translated; only those variables which were part of **termcap** will normally be output. Specifying the **–r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output. Mandatory padding is not supported. Because **termcap** strings are not as flexible, it is not always possible to convert a **terminfo** string capability into an equivalent **termcap** format. A subsequent conversion of the **termcap** file back into **terminfo** format will not necessarily reproduce the original **terminfo** source.

Some common **terminfo** parameter sequences, their **termcap** equivalents, and some terminal types which commonly have such sequences, are:

terminfo	**termcap**	**Representative Terminals**
%p1%c	**%.**	adm
%p1%d	**%d**	hp, ANSI standard, vt100
%p1%'x'%+%c	**%+x**	concept
%i	**%iq**	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	**%>xy**	concept
%p2 is printed before %p1	**%r**	hp

Use= Option [-u]

The **–u** option produces a **terminfo** source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the terminfo compiler *tic* does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a **use=** entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use=** fields that are superfluous. *infocmp* will flag any other *termname* **use=** fields that were not needed.

Other Options [-1eVv] [-F *file1 file2***] [-s d | i | l | c] [-w** *width***]**

–s Sorts the fields within each type according to the argument below:

- **d** Leave fields in the order that they are stored in the *terminfo* database.
- **i** Sort by *terminfo* name.
- **l** Sort by the long C variable name.
- **c** Sort by the *termcap* name.

If the **–s** option is not given, the fields printed out will be sorted alphabetically by the **terminfo** name within each type, except in the case of the **–C** or the **–L** options, which cause the sorting to be done by the **termcap** name or the long C variable name, respectively.

–F Compares terminfo files. This assumes that two following arguments are filenames. The files are searched for pairwise matches between entries, with two entries considered to match if any of their names do. The report printed to standard output lists entries with no matches in the other file, and entries with more than one match. For entries with exactly one match it includes a difference report.

–p Ignores padding specifications when comparing strings.

–v *n* Prints out tracing information on standard error as the program runs. Higher values of n induce greater verbosity.

–V Prints out the version of the program in use on standard error and exits.

–1 Causes the fields to be printed out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

–w Changes the output to *width* characters.

–R*subset* Restricts output to a given subset. This option is for use with archaic versions of terminfo like those on SVr1, Ultrix, or HP/UX that don't support the full set of SVR4/XSI Curses terminfo; and outright broken ports like AIX that have their own extensions incompatible with SVr4/XSI. Available terminfo subsets are **SVr1**, **Ultrix**, **HP**, and **AIX**; see *terminfo* for details.
You can also choose the subset "BSD" which selects only capabilities with termcap equivalents recognized by 4.4BSD.

–e Dumps the capabilities of the given terminal as a C initializer for a TERMTYPE structure (the terminal capability structure in the **<term.h>**). This option is useful for preparing versions of the curses library hardwired for a given terminal type.

Changing Databases [-i] [-A *directory***] [-B** *directory***]**

The location of the compiled **terminfo** database is taken from the environment variable *TERMINFO*. If the variable is not defined, or the terminal is not found in that location, the system **terminfo** database, usually in **$OPENNT_ROOT/usr/share/lib/terminfo**, will be used. The options **–A** and **–B** may be used to override this location. The **–A** option will set *TERMINFO* for the first *termname* and the **–B** option will set *TERMINFO* for the other *termname*s. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people.

–i Analyze the initialization (**is1**, **is2**, **is3**), and reset (**rs1**, **rs2**, **rs3**), strings in the entry. For each string, the code tries to analyze it into actions in terms of the other capabilities in the entry, certain X3.64/ISO 6429/ECMA-48 capabilities, and certain DEC VT-series private modes (the set of recognized special sequences has been selected for completeness over the existing terminfo database). Each report line consists of the capability name, followed by a colon and space, followed by a printable expansion of the capability string with sections matching recognized actions translated into {}-bracketed descriptions. Here is a list of the DEC/ANSI special sequences recognized:

Action	**Meaning**
RIS	full reset
SC	save cursor
RC	restore cursor
LL	home-down

RSR	reset scroll region
ISO DEC G0	enable DEC graphics for G0
ISO UK G0	enable UK chars for G0
ISO US G0	enable US chars for G0
ISO DEC G1	enable DEC graphics for G1
ISO UK G1	enable UK chars for G1
ISO US G1	enable US chars for G1
DECPAM	application keypad mode
DECPNM	normal keypad mode
DECANSI	enter ANSI mode
DEC[+-]CKM	application cursor keys
DEC[+-]ANM	set VT52 mode
DEC[+-]COLM	132-column mode
DEC[+-]SCLM	smooth scroll
DEC[+-]SCNM	reverse video mode
DEC[+-]OM	origin mode
DEC[+-]AWM	wraparound mode
DEC[+-]ARM	auto-repeat mode

It also recognizes a SGR action corresponding to ANSI/ISO 6429/ECMA Set Graphics Rendition, with the values NORMAL, BOLD, UNDERLINE, BLINK, and REVERSE. All but NORMAL may be prefixed with '+' (turn on) or '-' (turn off). An SGR0 designmates an empty highlight sequence (equivalent to {SGR:NORMAL}).

FILES

terminfo
Compiled terminal description database.

EXTENSIONS

The **–F** option is not supported in SVr4 curses. (It is primarily intended to help *infocmp*'s author, while wearing his terminfo/termcap maintainer hat, merge termcap/terminfo files from various sources into the master.)

The **–R**, **–p**, **–e** and **–i** options are not supported in SVr4 curses.

The **–r** option's notion of 'termcap' capabilities is System V Release 4's. Actual BSD curses versions will have a more restricted set. To see only the 4.4BSD set, use **–r –RBSD**.

BUGS

The **–F** option of *infocmp* should be a *toe* mode.

AUTHOR

Eric S. Raymond <esr@nark.thyrsus.com>

SEE

infocmp, *captoinfo*, *infotocap*, *tic*, *toe*

curses()

NAME

install – install binaries

SYNOPSIS

```
install [-cs] [-f flags] [-g group] [-m mode]
        [-o owner] file1 file2
install [-cs] [-f flags] [-g group] [-m mode]
        [-o owner] file1 ...  fileN directory
```

DESCRIPTION

The *install* utility moves the file(s) (or copies them, if the **–c** option is specified) to the target file or directory. If the destination is a directory, then the *file* is moved into *directory* with its original filename. If the target file already exists, it is overwritten if permissions allow.

–c Copy the file. This flag turns off the default behavior of *install* where it deletes the original file after creating the target.

–g Specify a group.

–m Specify an alternate mode. The default mode is set to rwxr-xr-x (0755). The specified mode may be either an octal or symbolic value; see *chmod* for a description of possible mode values.

–o Specify an owner.

–s *Install* exec's the command *strip* to strip binaries so that install can be portable over a large number of systems and binary types.

The *install* utility attempts to prevent moving a file onto itself.

Installing **/dev/null** creates an empty file.

DIAGNOSTICS

Upon successful completion a value of 0 is returned. Otherwise, a value of 1 is returned.

SEE ALSO

Revision 1.3 1997/11/12 17:18:36 SSI_DEV+johnmc # fixes required after proofing # *chgrp*, *chmod*, *cp*, *mv*, *strip*, *chown* # Revision 1.3 1997/11/12 17:18:36 SSI_DEV+johnmc # fixes required after proofing # # Revision 1.2 1997/11/02 16:16:32 SSI_DEV+johnmc # cleanup for alpha CD # # Revision 1.1 1997/10/21 14:24:41 SSI_DEV+johnmc # Initial revision # # Revision 1.1 1997/06/30 15:42:36 SSI_DEV+johnmc # Initial revision # # Revision 1.1 1997/06/30 15:42:36 SSI_DEV+johnmc # Initial revision

NAME

ispell – Interactive spelling checking

SYNOPSIS

```
ispell [common-flags] [-M|-N] [-L context] [-V] files

ispell [common-flags] -l

ispell [common-flags] [-f file] [-s] {-a|-A}

ispell [-d file] [-w chars] -c

ispell [-d file] [-w chars] -e [e]

ispell [-d file] -D

ispell -v [v]

common-flags:
 [-bBCmnPStx] [-d file] [-p file]
 [-w chars] [-W n] [-T type]
```

DESCRIPTION

Ispell is fashioned after the *spell* program from ITS (called *ispell* on Twenex systems.) The most common usage is "ispell filename". In this case, *ispell* will display each word which does not appear in the dictionary at the top of the screen and allow you to change it. If there are "near misses" in the dictionary (words which differ by only a single letter, a missing or extra letter, a pair of transposed letters, or a missing space or hyphen), then they are also displayed on following lines. As well as "near misses", ispell may display other guesses at ways to make the word from a known root, with each guess preceded by question marks. Finally, the line containing the word and the previous line are printed at the bottom of the screen. If your terminal can display in reverse video, the word itself is highlighted. You have the option of replacing the word completely, or choosing one of the suggested words. Commands are single characters as follows (case is ignored):

R	Replace the misspelled word completely.
Space	Accept the word this time only.
A	Accept the word for the rest of this *ispell* session.
I	Accept the word, capitalized as it is in the file, and update private dictionary.
U	Accept the word, and add an uncapitalized (actually, all lower-case) version to the private dictionary.
0-*n*	Replace with one of the suggested words.
L	Look up words in system dictionary (controlled by the WORDS compilation option).
X	Write the rest of this file, ignoring misspellings, and start next file.
Q	Exit immediately and leave the file unchanged.
!	Shell escape.
^L	Redraw screen.
^Z	Suspend ispell.
?	Give help screen.

"Normal" mode, as well as the **–l**, **–a**, and **–A** options (see below) also accepts the following "common" flags on the command line: **–B**, **–b**, **–C**, **–d**, **–m**, **–n**, **–p**, **–S**, **–T**, **–t**,

–W, **–w**, and **–x**. These options are described in more detail after the option list.

The *ispell* command takes the following options; when there are contradictory options, the last one on the command line takes precedence.

–A Read from standard input, usually a pipe, just like *a*, except that if a line begins with the string "&Include_File&", the rest of the line is taken as the name of a file to read for further words. Input returns to the original file when the include file is exhausted. Up to five levels of inclusion are allowed. If the environment variable *INCLUDE_STRING* is set, its value is the key string. (The ampersands, if any, must be included).

–a Read from standard input, usually a pipe. In this mode, *ispell* prints a one-line identifaction message and then reads lines of input. For each word in the line, *ispell* writes to standard output:

* The word was found in a dictionary.

+ root The word was found by affix removal.

– The word was found by compound formation (see **–C**).

& *word n c* : *nearmisses guesses*
The *word* was not in the dictionary, but there are near misses: *n* is the number of near misses; *c* is the number of characters between the beginning of the line and the misspelled word, and *list* is the list of near misses. Near misses are capitalized in the same manner as *word*, unless that capitalization is illegal. If the word could be made by adding illegal derivations to a root, there may also be a list of those *guesses*. See the next entry for the format of guesses.

? *word* 0 *c* : *guesses*
There are no near misses, but there are possible derivations. Each of these *guesses* is displayed as:

`[`*`prefix`*`+]`*`root`*`[-`*`prefix`*`][-`*`suffix`*`][+`*`suffix`*`]`

For example, "refries" would display as "re+fry-y+ies".

word c
The *word* is misspelled, there are no near misses or derivations, and the misspelled word occurs *c* characters from the beginning of the line.

–B Report run-together words with missing blanks as spelling errors. Contradicts **–C**.

–b Create a backup file by appending ".bak" to the name of the input file. Contradicts **–x**.

–C Consider run-together words as legal compounds. Contradicts **–B**.

–d *file* Specify an alternate dictionary file. For example, use **–d deutsch** to choose a German dictionary in a German installation.

–f *outfile*
Write results to a given *out*file rather than to standard output. Valid only with the **–a** or **–A** options.

–L *num* Use *num* as the number of lines of context to be shown at the bottom of the screen; the maximum value for *num* is subject to a system-imposed limit.

–l Produce a list of misspelled words from the standard input.

–M Display a one-line mini-menu at the bottom of the screen. Contradicts **–N**.

–m Make possible root / affix combinations that aren't in the dictionary. Contradicts **–P**.

–N Disply no mini-menu at the bottom of the screen.

–n The input file is in nroff / troff format. Contradicts **–t**.

–P Don't generate extra root / affix combinations. Contradicts **–m**.

–p *file* Specify an alternate personal dictionary.

–t The input file is in TeX or LaTeX format. Unless the **–n** or **–T** option is given, TeX mode is automatically used if the file being checked has a .tex extension. Contradicts **–n**.

–S Suppress *ispell*'s normal behavior of sorting the list of possible replacement words. Some people may prefer this, since it somewhat enhances the probability that the correct word will be low-numbered.

–T *type* Assume a given formatter type for all files, overriding the value determined from the file type. The *type* can either be a string or a file extension (defined in the dictionary's affix file); for English, they are **nroff**, **tex**, **.mm**, **.ms**, **.me**, **.man**, **.NeXT**, **.tex**, and **.bib**. The *type* can also be a supplied file suffix including the dot. **tex**, or it can be a supplied file extension. If no **–T** option appears and no type can be determined from the file name, the default string character type declared in the language affix file will be used.

–V Display characters not in the 7-bit ANSI printable character set so they are visible. Control characters prints as ^X for control-X; the delete character (octal 0177) prints as ^?. This is the format used by **cat –v**. Without this switch, *ispell* displays 8-bit characters "as is" if they have been defined as string characters for the chosen file type.

–v[**v**] Print version identification on the standard output and exit. If the switch is doubled, *ispell* prints the options that it was compiled with.

–W *n* Specify length of words that are always legal. The default value for *n* is 1. If you want all words to be checked against the dictionary, regardless of length, you might want to specify "**–W 0**." On the other hand, if your document specifies a lot of three-letter acronyms, you would specify "**–W 3**" to accept all words of three letters or less. Regardless of the setting of this option, *ispell* will only generate words that are in the dictionary as suggested replacements for words; this prevents the list from becoming too long. Obviously, this option can be very dangerous, since short misspellings may be missed. If you use this option a lot, you should probably make a last pass without it before you publish your document, to protect yourself against errors.

–w *chars* Specify additional characters that can be part of a word.

–x Don't create a backup file. Contradicts **–b**; the last one on the command line takes precedence.

–z Print to standard output all words not literally in the dictionary file. This is an INTERIX extension.

Nroff and Tex Modes

The **–n** and **–t** options select whether *ispell* runs in nroff/troff (**–n**) or TeX/LaTeX (**–t**) input mode. (The default is controlled by the DEFTEXFLAG installation option.)

In TeX/LaTeX mode, whenever a backslash ("\") is found, *ispell* will skip to the next whitespace or TeX/LaTeX delimiter. Certain commands contain arguments which should not be checked, such as labels and reference keys as are found in the \cite command, since they contain arbitrary, non-word arguments. Spell checking is also suppressed when in math mode. Thus, for example, given

```
\chapter {This is a Ckapter}
\cite{SCH86}
```

ispell will find "Ckapter" but not "SCH". The **–t** option does not recognize the TeX comment character "%", so comments are also spell-checked. It also assumes correct LaTeX syntax. Arguments to infrequently used commands and some optional arguments are sometimes checked unnecessarily. The bibliography will not be checked if *ispell* was compiled with IGNOREBIB defined. Otherwise, the bibliography will be checked but the reference key will not.

TeX/LaTeX mode is also automatically selected if an input file has the extension ".tex", unless overridden by the **–n** switch.

References for the *tib*(1) bibliography system, that is, text between a ''[.'' or ''<.'' and ''.]'' or ''.>'' will always be ignored in TeX/LaTeX mode.

Backup Files

The **–b** and **–x** options control whether *ispell* leaves a backup (.bak) file for each input file. The .bak file contains the pre-corrected text. If there are file opening / writing errors, the .bak file may be left for recovery purposes even with the **–x** option. The default for this option is controlled by the DEFNOBACKUPFLAG installation option.

Run-Together Words

The **–B** and **–C** options control how *ispell* handles run-together words, such as "notthe" for "not the". If **–B** is specified, such words will be considered as errors, and *ispell* will list variations with an inserted blank or hyphen as possible replacements. If **–C** is specified, run-together words will be considered to be legal compounds, so long as both components are in the dictionary, and each component is at least as long as a language-dependent minimum (3 characters, by default). This is useful for languages such as German and Norwegian, where many compound words are formed by concatenation. (Note that compounds formed from three or more root words will still be considered errors). The default for this option is language-dependent; in a multi-lingual installation the default may vary depending on which dictionary you choose.

Suggested Spellings (Guesses)

The **–P** and **–m** options control when *ispell* automatically generates suggested root / affix combinations for possible addition to your personal dictionary. (These are the entries in the "guess" list which are preceded by question marks.) If **–P** is specified, such guesses are displayed only if *ispell* cannot generate any possibilities that match the current dictionary. If **–m** is specified, such guesses are always displayed. This can be useful if the dictionary has a limited word list, or a word list with few suffixes. However, you should be careful when using this option, as it can generate guesses that produce illegal words. The default for this option is controlled by the dictionary file used.

Specifying Alternative Dictionary Files

The **–p** option is used to specify an alternate personal dictionary file. A personal dictionary file is simply a sorted list of words, one word to a line. If the file name does not begin with "/", the value of *HOME* is prefixed. Also, the shell variable *WORDLIST* may be set, which renames the personal dictionary in the same manner. The command line overrides any *WORDLIST* setting. If neither the **–p** switch nor the *WORDLIST* environment variable is given, *ispell* will search for a personal dictionary in both the current directory and **$HOME**, creating one in **$HOME** if none is found. The preferred name is constructed by appending ".ispell_" to the base name of the hash file. For example, if you use the English dictionary, your personal dictionary would be named ".ispell_english". However, if the file ".ispell_words" exists, it will be used as the personal dictionary regardless of the language hash file chosen. This feature is included primarily for backwards compatibility.

If the **–p** option is *not* specified, *ispell* will look for personal dictionaries in both the current directory and the home directory. If dictionaries exist in both places, they will be merged. If any words are added to the personal dictionary, they will be written to the current directory if a dictionary already existed in that place; otherwise they will be written to the dictionary in the home directory.

Specifying Non-Alphabetic Characters

The **–w** option may be used to specify characters other than alphabetics which may also appear in words. For instance, **–w** "&" will allow "AT&T" to be picked up. Underscores are useful in many technical documents. There is an admittedly crude provision in this option for 8-bit international characters. Non-printing characters may be specified in the usual way by inserting a backslash followed by the octal character code; e.g., "\014" for a form feed. Alternatively, if "n" appears in the character string, the (up to) three characters following are a DECIMAL code 0 – 255, for the character. For example, to include bells and form feeds in your words (an admittedly silly thing to do, but aren't most pedagogical examples):

```
n007n012
```

Numeric digits other than the three following "n" are simply numeric characters. Use of "n" does not conflict with anything because actual alphabetics have no meaning - alphabetics are already accepted. *Ispell* will typically be used with input from a file, meaning that preserving parity for possible 8 bit characters from the input text is OK. If you specify the **–l** option, and actually type text from the terminal, this may create problems if your stty settings preserve parity.

Piping Into ispell

The **–a** and **–A** options are intended to be used from other programs through a pipe. This mode is also suitable for interactive use when you want to figure out the spelling of a single word.

In this mode, *ispell* prints a one-line version identification message, and then begins reading lines of input. For each input line, a single line is written to the standard output for each word checked for spelling on the line. If the word was found in the main dictionary, or your personal dictionary, then the line contains only a '*'. If the word was found through affix removal, then the line contains a '+', a space, and the root word. If the word was found through compound formation (concatenation of two words, controlled by the **–C** option), then the line contains only a '–'.

If the word is not in the dictionary, but there are near misses, then the line contains an '&', a space, the misspelled word, a space, the number of near misses, the number of characters between the beginning of the line and the beginning of the misspelled word, a colon, another space, and a list of the near misses separated by commas and spaces. Following the near misses (and identified only by the count of near misses), if the word could be formed by adding (illegal) affixes to a known root, is a list of suggested derivations, again separated by commas and spaces. If there are no near misses at all, the line format is the same, except that the '&' is replaced by '?' (and the near-miss count is always zero). The suggested derivations following the near misses are in the form:

```
[prefix+] root [-prefix] [-suffix] [+suffix]
```

(e.g., "re+fry-y+ies" to get "refries") where each optional *pfx* and *sfx* is a string. Also, each near miss or guess is capitalized the same as the input word unless such capitalization is illegal; in the latter case each near miss is capitalized correctly according to the dictionary.

Finally, if the word does not appear in the dictionary, and there are no near misses, then the line contains a '#', a space, the misspelled word, a space, and the character offset from the beginning of the line. Each sentence of text input is terminated with an additional blank line, indicating that *ispell* has completed processing the input line.

These output lines can be summarized as follows:

OK	*
Root	+ <root>
Compound	–
Miss	& <original> <count> <offset>: <miss>, <miss>, ..., <guess>, ...
Guess	? <original> 0 <offset>: <guess>, <guess>, ...
None	# <original> <offset>

For example, a dummy dictionary containing the words "fray", "Frey", "fry", and "refried" might produce the following response to the command

```
$ echo 'frqy refries | ispell -a -m -d ./test.hash"
(#) International Ispell Version 3.0.05 (beta), 08/10/91
& frqy 3 0: fray, Frey, fry
& refries 1 5: refried, re+fry-y+ies
```

When in the **–a** and **–A** modes, *ispell* will also accept lines of single words prefixed with any of '*', '&', '@', '+', '-', '~', '#', '!', '%', or '^'. A line starting with '*' tells *ispell* to insert the word into the user's dictionary (similar to the I command). A line starting with '&'

tells *ispell* to insert an all-lowercase version of the word into the user's dictionary (similar to the U command). A line starting with '@' causes *ispell* to accept this word in the future (similar to the A command). A line starting with '+', followed immediately by **tex** or **nroff** will cause *ispell* to parse future input according the syntax of that formatter. A line consisting solely of a '+' will place *ispell* in TeX/LaTeX mode (similar to the **–t** option) and '-' returns *ispell* to nroff/troff mode (but these commands are obsolete). However, string character type is *not* changed; the '~' command must be used to do this. A line starting with '~' causes *ispell* to set internal parameters (in particular, the default string character type) based on the filename given in the rest of the line. (A file suffix is sufficient, but the period must be included. Instead of a file name or suffix, a unique name, as listed in the language affix file, may be specified.) However, the formatter parsing is *not* changed; the '+' command must be used to change the formatter. A line prefixed with '#' will cause the personal dictionary to be saved. A line prefixed with '!' will turn on *terse* mode (see below), and a line prefixed with '%' will return *ispell* to normal (non-terse) mode. Any input following the prefix characters '+', '-', '#', '!', or '%' is ignored, as is any input following the filename on a '~' line. To allow spell-checking of lines beginning with these characters, a line starting with '^' has that character removed before it is passed to the spell-checking code. It is recommended that programmatic interfaces prefix every data line with an uparrow to protect themselves against future changes in *ispell*.

To summarize these:

*	Add to personal dictionary
@	Accept word, but leave out of dictionary
#	Save current personal dictionary
~	Set parameters based on filename
+	Enter TeX mode
-	Exit TeX mode
!	Enter terse mode
%	Exit terse mode
^	Spell-check rest of line

In *terse* mode, *ispell* will not print lines beginning with '*', '+', or '–', all of which indicate correct words. This significantly improves running speed when the driving program is going to ignore correct words anyway.

The **–s** option is only valid in conjunction with the **–a** or **–A** options, and only on BSD-derived systems. If specified, *ispell* will stop itself with a SIGTSTP signal after each line of input. It will not read more input until it receives a SIGCONT signal. This may be useful for handshaking with certain text editors.

The **–f** option is only valid in conjunction with the **–a** or **–A** options.

Other Options

The **–c**, e**–[**1-4 and **–D** options of *ispell*, are primarily intended for use by the *munchlist* shell script. The **–c** switch causes a list of words to be read from the standard input. For each word, a list of possible root words and affixes will be written to the standard output. Some of the root words will be illegal and must be filtered from the output by other means; the *munchlist* script does this. As an example, the command:

```
$ echo BOTHER | ispell -c
BOTHER BOTHE/R BOTH/R
```

The **–e** switch is the reverse of **–c**; it expands affix flags to produce a list of words. For example, the command:

```
$ echo BOTH/R | ispell -e
BOTH BOTHER
```

An optional expansion level can also be specified. A level of 1 (**–e1**) is the same as **–e** alone. A level of 2 causes the original root / affix combination to be prepended to the line:

```
BOTH/R BOTH BOTHER
```

A level of 3 causes multiple lines to be output, one for each generated word, with the original root / affix combination followed by the word it creates:

```
BOTH/R BOTH
BOTH/R BOTHER
```

A level of 4 causes a floating-point number to be appended to each of the level-3 lines, giving the ratio between the length of the root and the total length of all generated words including the root:

```
BOTH/R BOTH 2.500000
BOTH/R BOTHER 2.500000
```

Finally, the **–D** flag causes the affix tables from the dictionary file to be dumped to standard output.

Capitalization

Unless your system administrator has suppressed the feature to save space, *ispell* is aware of the correct capitalizations of words in the dictionary and in your personal dictionary. As well as recognizing words that must be capitalized (e.g., George) and words that must be all-capitals (e.g., NASA), it can also handle words with "unusual" capitalization (e.g., "ITCorp" or "TeX"). If a word is capitalized incorrectly, the list of possibilities will include all acceptable capitalizations. (More than one capitalization may be acceptable; for example, my dictionary lists both "ITCorp" and "ITcorp".)

Normally, this feature will not cause you surprises, but there is one circumstance you need to be aware of. If you use "I" to add a word to your dictionary that is at the beginning of a sentence (e.g., the first word of this paragraph if "normally" were not in the dictionary), it will be marked as "capitalization required". A subsequent usage of this word without capitalization (e.g., the quoted word in the previous sentence) will be considered a misspelling by *ispell*, and it will suggest the capitalized version. You must then compare the actual spellings by eye, and then type "I" to add the uncapitalized variant to your personal dictionary. You can avoid this problem by using "U" to add the original word, rather than "I".

The rules for capitalization are as follows:

1. Any word may appear in all capitals, as in headings.
2. Any word that is in the dictionary in all-lowercase form may appear either in lowercase or capitalized (as at the beginning of a sentence).

3. Any word that has "funny" capitalization (i.e., it contains both cases and there is an uppercase character besides the first) must appear exactly as in the dictionary, except as permitted by rule (1). If the word is acceptable in all-lowercase, it must appear thus in a dictionary entry.

ENVIRONMENT

DICTIONARY
Default dictionary to use, if no **–d** flag is given.

WORDLIST
Personal dictionary file name.

INCLUDE_STRING
Code for file inclusion under the **–A** option.

FILES

$OPENNT_ROOT/usr/share/spell/british.hash
Hashed dictionary for British english.

$OPENNT_ROOT/usr/share/spell/english.hash
Hashed dictionary for American english.

$HOME/.ispell_*file*
User's private dictionary

.ispell_*file*
Directory-specific private dictionary

NOTES

The version of *ispell* supplied with INTERIX does not include the scripts and tools for building dictionary files. These are available from Softway Systems, Inc. upon request.

The original reference page lists these as bugs:

It takes several to many seconds for *ispell* to read in the hash table, depending on size.

When all options are enabled, *ispell* may take several seconds to generate all the guesses at corrections for a misspelled word; on slower machines this time is long enough to be annoying.

The hash table is stored as a quarter-megabyte (or larger) array, so a PDP-11 or 286 version does not seem likely.

Ispell should understand more *troff* syntax, and deal more intelligently with contractions.

Although small personal dictionaries are sorted before they are written out, the order of capitalizations of the same word is somewhat random.

When the **–x** flag is specified, *ispell* will unlink any existing .bak file.

There are too many flags, and many of them have non-mnemonic names.

AUTHOR

Pace Willisson (pace@mit-vax), 1983, based on the PDP-10 assembly version. That version was written by R. E. Gorin in 1971, and later revised by W. E. Matson (1974) and W. B. Ackerman (1978).

Collected, revised, and enhanced for the Usenet by Walt Buehring, 1987.

Table-driven multi-lingual version by Geoff Kuenning, 1987-88.

Large dictionaries provided by Bob Devine (vianet!devine).

A complete list of contributors is too large to list here, but is distributed with the ispell sources in the file "Contributors".

VERSION

The version of ispell described by this manual page is International Ispell Version 3.1.00, 10/08/93.

SEE ALSO

spell, *egrep*, *look*, *join*, *sort*,

NAME

join – relational database operator

SYNOPSIS

```
join [-a file_number | -v file_number] [ -e string]
     [-j file_number field] [-o list] [-t char]
     [-1 field] [-2 field] file1 file2
```

DESCRIPTION

The *join* utility performs an ‘‘equality join’’ on the specified files and writes the result to the standard output. The ‘‘join field’’ is the field in each file by which the files are compared. The first field in each line is used by default. There is one line in the output for each pair of lines in *file1* and *file2* which have identical join fields. Each output line consists of the join field, the remaining fields from *file1* and then the remaining fields from *file2*.

The default field separators are tab and space characters. In this case, multiple tabs and spaces count as a single field separator, and leading tabs and spaces are ignored. The default output field separator is a single space character.

Many of the options use file and field numbers. Both file numbers and field numbers are 1 based, i.e. the first file on the command line is file number 1 and the first field is field number 1. The following options are available:

–a *file_number*
: In addition to the default output, produce a line for each line in file *file_number* that cannot be paired.

–e *string*
: Replace empty output fields with *string*.

–o *list*
: The **–o** option specifies the fields that will be output from each file for each line with matching join fields. Each element of *list* has the form `file_number.field`, where *file_number* is a file number and *field* is a field number. The elements of list must be either comma (‘‘,’’) or whitespace separated. (The latter requires quoting to protect it from the shell, or, a simpler approach is to use multiple **–o** options.)

–t *char*
: Use character *char* as a field delimiter for both input and output. Every occurrence of *char* in a line is significant.

–v *file_number*
: Do not display the default output, but display a line for each line in file *file_number* that cannot be paired. The options **–v**1 and **–v**2 may be specified at the same time.

–1 *field*
: Join on the *field*’th field of file 1.

–2 *field*
: Join on the *field*’th field of file 2.

When the default field delimiter characters are used, the files to be joined should be ordered in the collating sequence of *sort*, using the **–b** option, on the fields on which they are to be joined, otherwise *join* may not report all field matches. When the field delimiter characters are specified by the **–t** option, the collating sequence should be the same as *sort* without the **–b** option.

If one of the arguments *file1* or *file2* is ‘‘-’’, the standard input is used.

DIAGNOSTICS

The *join* utility exits 0 on success, and >0 if an error occurs.

COMPATIBILITY

For compatibility with historic versions of *join*, the following options are available:

–a In addition to the default output, produce a line for each line in both *file1* and *file2* that cannot be paired.

–j1 *field* Join on the *field*’th field of file 1.

–j2 *field* Join on the *field*’th field of file 2.

–j *field* Join on the *field*’th field of both file 1 and file 2.

–o *list* ... Historical implementations of *join* permitted multiple arguments to the **–o** option. These arguments were of the form ‘‘file_number.field_number’’ as described for the current **–o** option. This has obvious difficulties in the presence of files named ‘‘1.2’’.

These options are available only so historic shell scripts don’t require modification and should not be used.

SEE ALSO

awk, *comm*, *cut*, *paste*, *sort*, *uniq*

NAME

kill – terminate or signal a process

SYNOPSIS

```
kill [-s signal_name] pid ...
kill [-signal_name] pid ...
kill [-signal_number] pid ...
kill [-l|-m]
```

DESCRIPTION

The *kill* utility sends the TERM signal to the processes specified by the *pid* operand(s).

Only a process with appropriate privileges may send signals to other users' processes.

The options are as follows:

–l List the signal names.

–m List the POSIX.2 signal number to signal name mapping.

–s *signal_name*
A symbolic signal name specifying the signal to be sent instead of the default TERM. The **–l** option displays the signal names.

–*signal_name*
Equivalent to **-s** *signal_name*.

–*signal_number*
A non-negative decimal integer, specifying the signal to be sent instead of the default TERM.

Some of the more commonly used signals:

1	SIGHUP	(hangup)
2	SIGINT	(interrupt)
3	SIGQUIT	(quit)
6	SIGABRT	(abort)
9	SIGKILL	(non-catchable, non-ignorable kill)
14	SIGALRM	(alarm clock)
15	SIGTERM	(software termination signal)

DIAGNOSTICS

The *kill* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

strsignal

NAME

ksh – Public domain KornShell

SYNOPSIS

```
ksh [±abCefhikmnprsuvxX] [±o option] [[–c command-string
    [command-name] | –s | file ] [argument ...]]
```

DESCRIPTION

ksh is a command interpreter that is intended for both interactive and shell script use. Its command language is a superset of the *sh* shell language.

The version here contains the features of the **ksh88** version, dated 11/16/88.

Shell Startup

The following options can be specified only on the command line:

–c *command-string*
: the shell executes the command(s) contained in *command-string*.

–i interactive mode — see below

–l login shell — see below interactive mode — see below

–s the shell reads commands from standard input; all non-option arguments are positional parameters

–r restricted mode — see below

In addition to the above, the options described in the **set** built-in command can also be used on the command line.

If neither the **–c** nor the **–s** options are specified, the first non-option argument specifies the name of a file the shell reads commands from; if there are no non-option arguments, the shell reads commands from standard input. The name of the shell (i.e., the contents of the **$0**) parameter is determined as follows: if the **–c** option is used and there is a non-option argument, it is used as the name; if commands are being read from a file, the file is used as the name; otherwise the name the shell was called with (i.e., argv[0]) is used.

A shell is *interactive* if the **–i** option is used or if both standard input and standard error are attached to a tty. An interactive shell has job control enabled (if available), ignores the INT, QUIT and TERM signals, and prints prompts before reading input (see *PS1* and *PS2* parameters). For non-interactive shells, the **trackall** option is on by default (see *set* command below).

A shell is **restricted** if the **–r** option is used or if either the basename of the name the shell is invoked with or the *SHELL* parameter is “rsh”. The following restrictions come into effect after the shell processes any profile and **$ENV** files:

- the **cd** command is disabled
- the *SHELL*, *ENV*, and *PATH* parameters can’t be changed
- command names can’t be specified with absolute or relative paths
- the **–p** option of the **command** built-in can’t be used
- redirections that create files can’t be used (i.e., >, >|, >>, <>)

A shell is **privileged** if the **–p** option is used. A privileged shell does not process *$HOME*/**.profile** nor the *ENV* parameter (see below), instead the file **$OPENNT_ROOT/etc/suid_profile** is processed.

If the basename of the name the shell is called with (i.e., argv[0]) starts with **–** or if the **–l** option is used, the shell is assumed to be a login shell and the shell reads and executes the contents of **$OPENNT_ROOT/etc/profile** and *$HOME*/**.profile** if they exist and are readable.

If the **ENV** parameter is set when the shell starts (or, in the case of login shells, after any profiles are processed), its value is subjected to parameter, command, arithmetic and tilde substitution and the resulting file (if any) is read and executed. If **ENV** parameter is not set (and not null) and *ksh* was compiled with the DEFAULT_ENV macro defined, the file named in that macro is included (after the above mentioned substitutions have been performed).

The exit status of the shell is 127 if the command file specified on the command line could not be opened, or non-zero if a fatal syntax error occurred during the execution of a script. In the absence of fatal errors, the exit status is that of the last command executed, or zero, if no command is executed.

Command Syntax

The shell begins parsing its input by breaking it into *word*s. Words, which are sequences of characters, are delimited by unquoted *white-space* characters (space, tab and newline) or *meta-characters* (**<**, **>**, **|**, **;**, **&**, **(** and **)**). Aside from delimiting words, spaces and tabs are ignored, while newlines usually delimit commands. The meta-characters are used in building the following tokens: **<**, **<&**, **<<**, **>**, **>&**, **>>**, etc. are used to specify redirections (see Input / Output Redirection below); **|** is used to create pipelines; **|&** is used to create co-processes (see Co-Processes below); **;** is used to separate commands; **&** is used to create asynchronous pipelines; **&&** and **||** are used to specify conditional execution; **;;** is used in **case** statements; **(** .. **)** are used to create subshells; and lastly, **((** .. **))** are used in arithmetic expressions;

White-space and meta-characters can be quoted individually using backslash (****), or in groups using double (") or single (**'**) quotes. Note that the following characters are also treated specially by the shell and must be quoted if they are to represent themselves: ****, ", **'**, **#**, **$**, **`**, **~**, **{**, **}**, *****, **?** and **[**. The first three of these are the above mentioned quoting characters (see Quoting below); **#**, if used at the beginning of a word, introduces a comment — everything after the **#** up to the nearest newline is ignored; **$** is used to introduce parameter, command and arithmetic substitutions (see Substitution below); **`** introduces an old-style command substitution (see Substitution below); **~** begins a directory expansion (see Tilde Expansion below); **{** and **}** delimit *csh* style alternations (see Brace Expansion below); and, finally, *****, **?** and **[** are used in file name generation (see File Name Patterns below).

As words and tokens are parsed, the shell builds commands, of which there are two basic types: *simple-commands*, typically programs that are executed, and *compound-commands*, such as **for** and **if** statements, grouping constructs and function definitions.

A simple-command consists of some combination of parameter assignments (see Parameters below), input / output redirections (see Input / Output Redirections below), and command words; the only restriction is that parameter assignments come before any command words. The command words, if any, define the command that is to be executed and its arguments. The command may be a shell built-in command, a function

or an *external command*, i.e., a separate executable file that is located using the *PATH* parameter (see Command Execution below). Note that all command constructs have an *exit status*: for external commands, this is related to the status returned by *wait*( ); the exit status of other command constructs (built-in commands, functions, compound-commands, pipelines, lists, etc.) are all well defined and are described where the construct is described. The exit status of a command consisting only of parameter assignments is that of the last command substitution performed during the parameter assignment or zero if there were no command substitutions.

Commands can be chained together using the **|** token to form *pipelines*, in which the standard output of each command but the last is piped (see *pipe*( )) to the standard input of the following command. The exit status of a pipeline is that of its last command. A pipeline may be prefixed by the **!** reserved word which causes the exit status of the pipeline to be logically complemented: if the original status was 0 the complemented status will be 1, and if the original status was not 0, then the complemented status will be 0.

Lists of commands can be created by separating pipelines by any of the following tokens: **&&**, **||**, **&**, **|&** and **;**. The first two are for conditional execution: *cmd1* **&&** *cmd2* executes *cmd2* only if the exit status of *cmd1* is zero; **||** is the opposite — *cmd2* is executed only if the exit status of *cmd1* is non-zero. **&&** and **||** have equal precedence which is higher than that of **&**, **|&** and **;**, which also have equal precedence. The **&** token causes the preceding command to be executed asynchronously, that is, the shell starts the command, but does not wait for it to complete (the shell does keep track of the status of asynchronous commands — see "Job Control" below). When an asynchronous command is started when job control is disabled (i.e., in most scripts), the command is started with signals INT and QUIT ignored and with input redirected from **/dev/null** (however, redirections specified in the asynchronous command have precedence). The **|&** operator starts a *co-process* which is special kind of asynchronous process (see Co-Processes below). Note that a command must follow the **&&** and **||** operators, while a command need not follow **&**, **|&** and **;**. The exit status of a list is that of the last command executed, with the exception of asynchronous lists, for which the exit status is 0.

Compound commands are created using the following reserved words — these words are only recognized if they are unquoted and if they are used as the first word of a command (i.e., they can't be preceded by parameter assignments or redirections):

case	**else**	**function**	**then**	**!**
do	**esac**	**if**	**time**	**[[**
done	**fi**	**in**	**until**	**{**
elif	**for**	**select**	**while**	**}**

Note: Some shells (but not this one) execute control structure commands in a subshell when one or more of their file descriptors are redirected, so any environment changes inside them may fail. To be portable, the **exec** statement should be used instead to redirect file descriptors before the control structure.

In the following compound command descriptions, command lists (denoted as *list*) that are followed by reserved words must end with a semi-colon, a newline or a (syntactically correct) reserved word. For example:

```
{ echo foo; echo bar; }
{ echo foo; echo bar<newline>}
{ { echo foo; echo bar; } }
```

are all valid, but

```
{ echo foo; echo bar }
```

is not.

(*list* **)** Execute *list* in a subshell. There is no implicit way to pass environment changes from a subshell back to its parent.

{ *list* **}** Compound construct; *list* is executed, but not in a subshell. Note that **{** and **}** are reserved words, not meta-characters.

case *word* **in** [[**(**] *pattern* [**|** *pattern*] ... **)** *list* **;;**] ... **esac**

The **case** statement attempts to match *word* against the specified *pattern*s; the *list* associated with the first successfully matched pattern is executed. Patterns used in **case** statements are the same as those used for file name patterns except that the restrictions regarding **.** and **/** are dropped. Note that any unquoted space before and after a pattern is stripped; any space with a pattern must be quoted. Both the word and the patterns are subject to parameter, command, and arithmetic substitution as well as tilde substitution.

For historical reasons, open and close braces may be used instead of **in** and **esac** (e.g., **case $foo { *) echo bar; }**). The exit status of a **case** statement is that of the executed *list*; if no *list* is executed, the exit status is zero.

for *name* [**in** *word ... term*] **do** *list* **done**

Where *term* is either a newline or a **;**. For each *word* in the specified word list, the parameter *name* is set to the word and *list* is executed. If **in** is not used to specify a word list, the positional parameters ("**$1**", "**$2**", etc.) are used instead. For historical reasons, open and close braces may be used instead of **do** and **done** (e.g., **for i; { echo $i; }**). The exit status of a **for** statement is the last exit status of *list*; if *list* is never executed, the exit status is zero.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

If the exit status of the first *list* is zero, the second *list* is executed; otherwise the *list* following the **elif**, if any, is executed with similar consequences. If all the lists following the **if** and **elif**s fail (i.e., exit with non-zero status), the *list* following the **else** is executed. The exit status of an **if** statement is that of non-conditional *list* that is executed; if no non-conditional *list* is executed, the exit status is zero.

select *name* [**in** *word ... term*] **do** *list* **done**

Where *term* is either a newline or a **;**. The **select** statement provides an automatic method of presenting the user with a menu and selecting from it. An enumerated list of the specified *words* are printed on standard error, followed by a prompt (*PS3*, normally '**#?** '). A number corresponding to one of the enumerated words is then read from standard input, *name* is set to the selected word (or is unset if the selection is not valid), *REPLY* is set to what was read (leading / trailing space is stripped), and *list* is executed. If a blank line (that is, zero or more *IFS* characters) is entered, the menu is re-printed without executing *list*. When *list* completes, the enumerated list is printed if *REPLY* is null, the prompt is printed, and so on. This process is repeated until an end-of-file is read, an interrupt is received or a break statement is executed inside the loop. If **in** *word* **...** is omitted, the positional parameters are used (i.e., "**$1**", "**$2**", etc.). For historical reasons, open and close braces may be used instead of

do and **done** (e.g., **select i; { echo $i; }**). The exit status of a **select** statement is zero if a **break** statement is used to exit the loop, non-zero otherwise.

time *pipeline*

The shell executes the pipeline and writes to standard error the elapsed time, the user time, and the system time for the command.

Redirections after the *pipeline* affect the last command in the pipeline, not the *time* command.

until *list* **do** *list* **done**

This works like **while**, except that the body is executed only while the exit status of the first *list* is non-zero.

while *list* **do** *list* **done**

A **while** is a prechecked loop. Its body is executed as often as the exit status of the first *list* is zero. The exit status of a **while** statement is the last exit status of the *list* in the body of the loop; if the body is not executed, the exit status is zero.

function *name* **{** *list* **}**

Defines the function *name*. See "Functions" below. Note that redirections specified after a function definition are performed whenever the function is executed, not when the function definition is executed.

name **()** *command*

Mostly the same as **function**. See "Functions" below.

((*expression* **))**

The arithmetic expression *expression* is evaluated; equivalent to **let** "*expression*". See Arithmetic Expressions and the **let** command below.

[[*expression* **]]**

Similar to the **test** and **[...]** commands (described later), with the following exceptions:

- Field splitting and file name generation are not performed on arguments.
- The **–a** (and) and **–o** (or) operators are replaced with **&&** and **||**, respectively.
- Operators (e.g. **–f**, **=**, **!**, etc.) must be unquoted.
- The second operand of **!=** and **=** expressions are patterns (e.g., the comparison in

  ```
  [[ foobar = f*r ]]
  ```

  succeeds).
- There are two additional binary operators: **<** and **>** which return true if their first string operand is less than, or greater than, their second string operand, respectively.
- The single argument form of **test**, which tests if the argument has non-zero length, is not valid — explicit operators must be always be used, e.g., instead of

  ```
  [str]
  ```

  use

  ```
  [[–n str]]
  ```

- Parameter, command and arithmetic substitutions are performed as expressions are evaluated and lazy expression evaluation is used for the **&&** and || operators. This means that in the statement

  ```
  [[ -r foo && $(< foo) = b*r ]]
  ```

  the **$(< foo)** is evaluated if and only if the file **foo** exists and is readable.

Quoting

Quoting is used to prevent the shell from treating characters or words specially. There are three methods of quoting: First, \ quotes the following character, unless it is at the end of a line, in which case both the \ and the newline are stripped. Second, a single quote (') quotes everything up to the next single quote (this may span lines). Third, a double quote (") quotes all characters, except **$**, **`** and \, up to the next unquoted double quote. **$** and **`** inside double quotes have their usual meaning (i.e., parameter, command or arithmetic substitution) except no field splitting is carried out on the results of double-quoted substitutions. If a \ inside a double-quoted string is followed by \, **$**, **`** or ", it is replaced by the second character; if it is followed by a newline, both the \ and the newline are stripped; otherwise, both the \ and the character following are unchanged.

Note: see "POSIX Mode" below for a special rule regarding sequences of the form "...`...\"...`...".

Aliases

There are two types of aliases: normal command aliases and tracked aliases. Command aliases are normally used as a short hand for a long or often used command. The shell expands command aliases (i.e., substitutes the alias name for its value) when it reads the first word of a command. An expanded alias is re-processed to check for more aliases. If a command alias ends in a space or tab, the following word is also checked for alias expansion. The alias expansion process stops when a word that is not an alias is found, when a quoted word is found or when an alias word that is currently being expanded is found.

The following command aliases are defined automatically by the shell:

```
autoload='typeset -fu'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
local='typeset'
nohup='nohup '
r='fc -e -'
stop='kill -STOP'
suspend='kill -STOP $$'
type='whence -v'
```

Tracked aliases allow the shell to remember where it found a particular command. The first time the shell does a path search for a command that is marked as a tracked alias, it saves the full path of the command. The next time the command is executed, the shell checks the saved path to see that it is still valid, and if so, avoids repeating the path search. Tracked aliases can be listed and created using **alias –t**. Note that changing the *PATH* parameter clears the saved paths for all tracked aliases. If the **trackall** option is

set (i.e., **set –o trackall** or **set –h**), the shell tracks all commands. This option is set automatically for non-interactive shells. For interactive shells, only the following commands are automatically tracked: *cat*, *cc*, *chmod*, *cp*, *date*, *ed*, *emacs*, *grep*, *ls*, *mail*, *make*, *mv*, *pr*, *rm*, *sed*, *sh*, *vi* and *who*. (Note that *emacs* is not currently part of any INTERIX product, and that *cc* is available only in the Software Development Kit.)

Substitution

The first step the shell takes in executing a simple-command is to perform substitutions on the words of the command. There are three kinds of substitution: parameter, command and arithmetic. Parameter substitutions, which are described in detail in the next section, take the form **$name** or **${**...**}**; command substitutions take the form **$(***command***)** or **‘***command***‘**; and arithmetic substitutions take the form **$((***expression***))**.

If a substitution appears outside of double quotes, the results of the substitution are generally subject to word or field splitting according to the current value of the *IFS* parameter. The *IFS* parameter specifies a list of characters which are used to break a string up into several words; any characters from the set space, tab and newline that appear in the IFS characters are called *IFS white space*. Sequences of one or more IFS white space characters, in combination with zero or one non-IFS white space characters delimit a field. As a special case, leading and trailing IFS white space is stripped (i.e., no leading or trailing empty field is created by it); leading or trailing non-IFS white space does create an empty field. Example: if *IFS* is set to ‘<space>:’, the sequence of characters ‘<space>A<space>:<space><space>B::D’ contains four fields: ‘A’, ‘B’, ‘’ and ‘D’. Note that if the *IFS* parameter is set to the null string, no field splitting is done; if the parameter is unset, the default value of space, tab and newline is used.

The results of substitution are, unless otherwise specified, also subject to brace expansion and file name expansion (see the relevant sections below).

A command substitution is replaced by the output generated by the specified command, which is run in a subshell. For **$(***command***)** substitutions, normal quoting rules are used when *command* is parsed, however, for the **‘***command***‘** form, a \ followed by any of **$**, **‘** or \ is stripped (a \ followed by any other character is unchanged). As a special case in command substitutions, a command of the form **<** *file* is interpreted to mean substitute the contents of *file* ($(< foo) has the same effect as $(cat foo), but it is carried out more efficiently because no process is started).

Note: $(*command***)** expressions are currently parsed by finding the matching parenthesis, regardless of quoting. This will hopefully be fixed soon.

Arithmetic substitutions are replaced by the value of the specified expression. For example, the command **echo $((2+3*4))** prints 14. See Arithmetic Expressions for a description of an *expression*.

Parameters

Parameters are shell variables; they can be assigned values and their values can be accessed using a parameter substitution. A parameter name is either one of the special single punctuation or digit character parameters described below, or a letter followed by zero or more letters or digits (‘_’ counts as a letter). Parameter substitutions take the form **$***name* or **${***name***}**, where *name* is a parameter name. If substitution is performed on a parameter that is not set, a null string is substituted unless the **nounset** option (**set –o nounset** or **set –u**) is set, in which case an error occurs.

Parameters can be assigned values in a number of ways. First, the shell implicitly sets some parameters like **#**, *PWD*, etc.; this is the only way the special single character parameters are set. Second, parameters are imported from the shell's environment at startup. Third, parameters can be assigned values on the command line, for example, '**FOO=bar**' sets the parameter FOO to bar; multiple parameter assignments can be given on a single command line and they can be followed by a simple-command, in which case the assignments are in effect only for the duration of the command (such assignments are also exported, see below for implications of this). Note that both the parameter name and the **=** must be unquoted for the shell to recognize a parameter assignment. The fourth way of setting a parameter is with the **export**, **readonly** and **typeset** commands; see their descriptions in the "Command Execution" section. Fifth, **for** and **select** loops set parameters as well as the **getopts**, **read** and **set –A** commands. Lastly, parameters can be assigned values using assignment operators inside arithmetic expressions (see Arithmetic Expressions below) or using the **${***name***=***value***}** form of parameter substitution (see below).

Parameters with the export attribute (set using the **export** or **typeset –x** commands, or by parameter assignments followed by simple commands) are put in the environment (see *environ*(5)) of commands run by the shell as *name*=*value* pairs. The order in which parameters appear in the environment of a command is unspecified. When the shell starts up, it extracts parameters and their values from its environment and automatically sets the export attribute for those parameters.

Modifiers can be applied to the **${***name***}** form of parameter substitution:

${*name***:-***word***}**
: If *name* is set and not null, it is substituted, otherwise *word* is substituted.

${*name***:+***word***}**
: If *name* is set and not null, *word* is substituted, otherwise nothing is substituted.

${*name***:=***word***}**
: If *name* is set and not null, it is substituted, otherwise it is assigned *word* and the resulting value of *name* is substituted.

${*name***:?***word***}**
: If *name* is set and not null, it is substituted, otherwise *word* is printed on standard error (preceded by *name*:) and an error occurs (normally causing termination of a shell script, function or .-script). If word is omitted the string 'parameter null or not set' is used instead.

In the above modifiers, the **:** can be omitted, in which case the conditions only depend on *name* being set (as opposed to set and not null). If *word* is needed, parameter, command, arithmetic and tilde substitution are performed on it; if *word* is not needed, it is not evaluated.

The following forms of parameter substitution can also be used:

${#*name***}**
: The number of positional parameters if *name* is *****, **@** or is not specified, or the length of the string value of parameter *name*.

${#***name***[*]}**, **${#*name***[@]}**
: The number of elements in the array *name*.

${***name***#***pattern***}**, **${*name***##** *pattern***}**
: If *pattern* matches the beginning of the value of parameter *name*, the matched text is deleted from the result of substitution. A single **#** results in the shortest match, two **#**'s results in the longest match.

${***name***%***pattern***}**, **${*name***%%***pattern***}**
Like **${..#..}** substitution, but it deletes from the end of the value.

The following special parameters are implicitly set by the shell and cannot be set directly using assignments:

! Process id of the last background process started. If no background processes have been started, the parameter is not set.
The number of positional parameters (i.e., **$1**, **$2**, etc.).
$ The process ID of the shell, or the PID of the original shell if it is a subshell.
– The concatenation of the current single letter options (see **set** command below for list of options).
? The exit status of the last non-asynchronous command executed. If the last command was killed by a signal, **$?** is set to 128 plus the signal number.
0 The name the shell was invoked with (i.e., **argv[0]**), or the **command-name** if it was invoked with the **–c** option and the **command-name** was supplied, or the *file* argument, if it was supplied. If the **posix** option is not set, **$0** is the name of the current function or script.
1 ... **9** The first nine positional parameters that were supplied to the shell, function or **.**-script. Further positional parameters may be accessed using **${***number***}**.
***** All positional parameters (except parameter 0), i.e., **$1 $2 $3**.... If used outside of double quotes, parameters are separate words (which are subjected to word splitting); if used within double quotes, parameters are separated by the first character of the *IFS* parameter (or the empty string if *IFS* is null).
@ Same as **$***, unless it is used inside double quotes, in which case a separate word is generated for each positional parameter — if there are no positional parameters, no word is generated ("$@" can be used to access arguments, verbatim, without loosing null arguments or splitting arguments with spaces).

The following parameters are set and / or used by the shell:

_ (underscore)
In interactive use, this parameter is set to the last word of the previous command. When a command is executed, this parameter is set to the full path of the command and is placed in the command's environment. When *MAILPATH* messages are evaluated, this parameter contains the name of the file that changed (see *MAILPATH* parameter below).
CDPATH
Search path for the *cd* built-in command. Works the same way as *PATH* for those directories not beginning with **/** in *cd* commands. Note that if *CDPATH* is set and does not contain **.** nor an empty path, the current directory is not searched.
COLUMNS
Set to the number of columns on the terminal or window. Currently set to the **cols** value as reported by *stty* if that value is non-zero. This parameter is used by the interactive line editing modes, and by **select**, **set –o** and **kill –l** commands to format information in columns.
EDITOR
If the *VISUAL* parameter is not set, this parameter controls the command line editing mode for interactive shells. See *VISUAL* parameter below for how this works.
ENV If this parameter is found to be set after any profile files are executed, the expanded value is used as a shell start-up file. It typically contains function

and alias definitions.

ERRNO Integer value of the shell's *errno* variable — indicates the reason the last system call failed.

EXECSHELL
If set, this parameter is assumed to contain the shell that is to be used to execute commands that *execve*( ) fails to execute and which do not start with a '**#!** *shell*' sequence.

FCEDIT The editor used by the **fc** command (see below).

FPATH Like *PATH*, but used when an undefined function is executed to locate the file defining the function. It is also searched when a command can't be found using *PATH*. See "Functions" below for more information.

HISTFILE
The name of the file used to store history. When assigned to, history is loaded from the specified file. Also, several invocations of the shell running on the same machine will share history if their *HISTFILE* parameters all point at the same file.

Note: if *HISTFILE* isn't set, no history file is used. This is different from the original KornShell, which uses **$HOME/.sh_history**; in future, *ksh* may also use a default history file.

HISTSIZE
The number of commands normally stored for history, default 128.

HOME The default directory for the **cd** command and the value substituted for an unqualified **~** (see Tilde Expansion below).

IFS Internal field separator, used during substitution and by the **read** command, to split values into distinct arguments; normally set to space, tab and newline. See "Substitution" above for details.

Note: this parameter is not imported from the environment when the shell is started.

KSH_VERSION
The version of shell and the date the version was created (readonly). See also the version commands in Emacs Interactive Input Line Editing and Vi Interactive Input Line Editing, below.

LINENO
The line number of the function or shell script that is currently being executed.

LINES Set to the number of lines on the terminal or window.

MAIL If set, the user will be informed of the arrival of mail in the named file. This parameter is ignored if the *MAILPATH* parameter is set.

MAILCHECK
How often, in seconds, the shell will check for mail in the file(s) specified by *MAIL* or *MAILPATH*. If 0, the shell checks before each prompt. The default is 600 (10 minutes).

MAILPATH
A list of files to be checked for mail. The list is colon separated, and each file may be followed by a **?** and a message to be printed if new mail has arrived. Command, parameter and arithmetic substitution is performed on the message, and, during substitution, the parameter **$_** contains the name of the file. The default message is **`you have mail in $_`**.

OLDPWD
The previous working directory. Unset if **cd** has not successfully changed directories since the shell started, or if the shell doesn't know where it is.

OPTARG
When using **getopts**, it contains the argument for a parsed option, if it requires one.

OPTIND
The index of the last argument processed when using **getopts**. Assigning 1 to this parameter causes **getopts** to process arguments from the beginning the next time it is invoked.

PATH A colon separated list of directories that are searched when looking for commands and **.**'d files. An empty string resulting from a leading or trailing colon, or two adjacent colons is treated as a '.', the current directory.

POSIXLY_CORRECT
If set, this parameter causes the **posix** option to be enabled. See "POSIX Mode" below.

PPID The process ID of the shell's parent (readonly).

PS1 *PS1* is the primary prompt for interactive shells. Parameter, command and arithmetic substitutions are performed, and **!** is replaced with the current command number (see **fc** command below). Default is '**$** '.

PS2 Secondary prompt string, by default '**>** ', used when more input is needed to complete a command.

PS3 Prompt used by **select** statement when reading a menu selection. Default is '**#?** '.

PS4 Used to prefix commands that are printed during execution tracing (see **set –x** command below). Parameter, command and arithmetic substitutions are performed before it is printed. Default is '**+** '.

PWD The current working directory. Maybe unset or null if shell doesn't know where it is.

RANDOM
A simple random number generator. Every time *RANDOM* is referenced, it is assigned the next number in a random number series. The point in the series can be set by assigning a number to *RANDOM* (see *rand*( )).

REPLY Default parameter for the **read** command if no names are given. Also used in **select** loops to store the value that is read from standard input.

SECONDS
The number of seconds since the shell started or, if the parameter has been assigned an integer value, the number of seconds since the assignment plus the value that was assigned.

TMOUT
If set to a positive integer in an interactive shell, it specifies the maximum number of seconds the shell will wait for input after printing the primary prompt (*PS1*). If the time is exceeded, the shell exits.

TMPDIR
The directory shell temporary files are created in. If this parameter is not set, or does not contain the absolute path of a writable directory, temporary files are created in **$OPENNT_ROOT/tmp**. (The root of the directory depends upon where the INTERIX distribution is installed.)

VISUAL
If set, this parameter controls the command line editing mode for interactive shells. If the last component of the path specified in this parameter contains the string **vi**, **emacs** or **gmacs**, the vi, emacs or gmacs (Gosling emacs) editing mode is enabled, respectively.

Tilde Expansion

Tilde expansion, which is done in parallel with parameter substitution, is done on words starting with an unquoted **~**. The characters following the tilde, up to the first **/**, if any, are assumed to be a login name. If the login name is empty, **+** or **–**, the value of the *HOME*, *PWD*, or *OLDPWD* parameter is substituted, respectively. Otherwise, the password file is searched for the login name, and the tilde expression is substituted with the user's home directory. If the login name is not found in the password file or if any quoting or parameter substitution occurs in the login name, no substitution is performed.

In parameter assignments (those preceding a simple-command or those occurring in the arguments of **alias**, **export**, **readonly**, and **typeset**), tilde expansion is done after any unquoted colon (**:**), and login names are also delimited by colons.

The home directory of previously expanded login names are cached and re-used. The **alias –d** command may be used to list, change and add to this cache (e.g., 'alias –d fac=/ / C / Interix / local / facilities; cd ~fac / bin').

On a Windows NT system, tilde expansion depends upon the contents of the *local* password database. If the login name comes from another domain (even if you're logged into that domain), it's not checked for information, and the tilde isn't properly expanded. You can use the **alias –d** command to add login names, as already mentioned.

Brace Expansion (alternation)

Brace expressions, which take the form *prefix***{***str1***,...,***strN***}***suffix* are expanded to N words, each of which is the concatenation of *prefix*, *str*i and *suffix* (e.g., 'a{c,b{X,Y},d}e' expands to four word: ace, abXe, abYe, and ade). As noted in the example, brace expressions can be nested and the resulting words are not sorted. Brace expressions must contain an unquoted comma (**,**) for expansion to occur (i.e., **{}** and **{foo}** are not expanded). Brace expansion is carried out after parameter substitution and before file name generation.

File Name Patterns

A file name pattern is a word containing one or more unquoted **?** or ***** characters or **[..]** sequences. Once brace expansion has been performed, the shell replaces file name patterns with the sorted names of all the files that match the pattern (if no files match, the word is left unchanged). The pattern elements have the following meaning:

? Matches any single character.

***** Matches any sequence of characters.

[..] Matches any of the characters inside the brackets. Ranges of characters can be specified by separating two characters by a **–**, e.g., **[a0–9]** matches the letter **a** or any digit. In order to represent itself, a **–** must either be quoted or the first or last character in the character list. Similarly, a **]** must be quoted or the first character in the list if it is represent itself instead of the end of the list. Also, a **!** appearing at the start of the list has special meaning (see below), so to represent itself it must be quoted or appear later in the list.

[!..] like **[..]**, except it matches any character not inside the brackets.

(pattern **|** ... **|***pattern***)**
Matches any string of characters that matches zero or more occurences of the specified patterns. Example: the pattern ***(foo | bar)** matches the strings '', 'foo', 'bar', 'foobarfoo', etc..

+(*pattern* **|** ... **|***pattern***)**
Matches any string of characters that matches one or more occurences of the specified patterns. Example: the pattern **+(foo | bar)** matches the strings 'foo', 'bar', 'foobarfoo', etc..

?(*pattern* **|** ... **|***pattern***)**
Matches the empty string or a string that matches one of the specified patterns. Example: the pattern **?(foo | bar)** only matches the strings '', 'foo' and 'bar'.

@(*pattern* **|** ... **|***pattern***)**
Matches a string that matches one of the specified patterns. Example: the pattern **@(foo | bar)** only matches the strings 'foo' and 'bar'.

!(*pattern* **|** ... **|***pattern***)**
Matches any string that does not match one of the specified patterns. Examples: the pattern **!(foo | bar)** matches all strings except 'foo' and 'bar'; the pattern **!(*)** matches no strings; the pattern **!(?)*** matches all strings (think about it).

Note that *ksh* currently never matches **.** and **..**, but the original ksh, Bourne sh and bash do, so this may have to change (too bad).

Note that none of the above pattern elements match either a period (**.**) at the start of a file name or a slash (*/*), even if they are explicitly used in a **[..]** sequence; also, the names **.** and **..** are never matched, even by the pattern **.***.

If the **markdirs** option is set, any directories that result from file name generation are marked with a trailing **/**.

The POSIX character classes (i.e., **[:***class-name***:]** inside a **[..]** expression) are not yet implemented.

Input/Output Redirection

When a command is executed, its standard input, standard output and standard error (file descriptors 0, 1 and 2, respectively) are normally inherited from the shell. Three exceptions to this are commands in pipelines, for which standard input and / or standard output are those set up by the pipeline, asynchronous commands created when job control is disabled, for which standard input is initially set to be from **/dev/null**, and commands for which any of the following redirections have been specified:

> *file* Standard output is redirected to *file*. If *file* does not exist, it is created; if it does exist, is a regular file and the **noclobber** option is set, an error occurs, otherwise the file is truncated. Note that this means the command *cmd* < *foo* > *foo* will open *foo* for reading and then truncate it when it opens it for writing, before *cmd* gets a chance to actually read *foo*.

>| *file* Same as **>**, except the file is truncated, even if the **noclobber** option is set.

>> *file* Same as **>**, except the file an existing file is appended to instead of being truncated. Also, the file is opened in append mode, so writes always go to the end of the file (see *open*( )).

< *file* Standard input is redirected from *file*, which is opened for reading.

<> *file* Same as **<**, except the file is opened for reading and writing.

<< *marker*
After reading the command line containing this kind of redirection (called a here document), the shell copies lines from the command source into a temporary file until a line matching *marker* is read. When the command is executed, standard input is redirected from the temporary file. If *marker*

contains no quoted characters, the contents of the temporary file are processed as if enclosed in double quotes each time the command is executed, so parameter, command and arithmetic substitutions are performed, along with backslash (\) escapes for **$**, **‘**, \ and **\newline**. If multiple here documents are used on the same command line, they are saved in order.

<<- *marker*
: Same as **<<**, except leading tabs are stripped from lines in the here document.

<& *fd*
: Standard input is duplicated from file descriptor *fd*. *fd* can be a single digit, indicating the number of an existing file descriptor, the letter **p**, indicating the file descriptor associated with the output of the current co-process, or the character **–**, indicating standard input is to be closed.

>& *fd*
: same as **<&**, except the operation is done on standard output.

In any of the above redirections, the file descriptor that is redirected (i.e., standard input or standard output) can be explicitly given by preceding the redirection with a single digit. Parameter, command and arithmetic substitutions, tilde substitutions and file name generation are all performed on the *file*, *marker* and *fd* arguments of redirections. Note however, that the results of any file name generation are only used if a single file is matched; if multiple files match, the word with the unexpanded file name generation characters is used. Note that in restricted shells, redirections which can create files cannot be used.

For simple-commands, redirections may appear anywhere in the command, for compound-commands (**if** statements, etc.), any redirections must appear at the end. Redirections are processed after pipelines are created and in the order they are given, so

```
cat /foo/bar 2>&1 > /dev/null | cat –n
```

will print an error with a line number prepended to it.

Arithmetic Expressions

Integer arithmetic expressions can be used with the **let** command, inside **$((..))** expressions, inside array references (e.g., *name***[***expr***]**), as numeric arguments to the **test** command, and as the value of an assignment to an integer parameter.

Expression may contain alpha-numeric parameter identifiers, array references, and integer constants and may be combined with the following C operators (listed and grouped in increasing order of precedence).

Unary operators:+ – ! ~ ++ ––
Binary operators:,
= *= /= %= += –= <<= >>= &= ^= |=
||
&&
|
^
&
== !=
< <= >= >

<< >>
+ –
* / %
Ternary operator:?:
(precedence is immediately higher than assignment)
Grouping operators:( )

Integer constants may be specified with arbitrary bases using the notation *base***#***number*, where *base* is a decimal integer specifying the base, and *number* is a number in the specified base.

The operators are evaluated as follows:

unary **+** Result is the argument (included for completeness).
unary **–** Negation.
! Logical not; the result is 1 if argument is zero, 0 if not.
~ Arithmetic (bit-wise) not.
++ Increment; must be applied to a parameter (not a literal or other expression) — the parameter is incremented by 1. When used as a prefix operator, the result is the incremented value of the parameter; when used as a postfix operator, the result is the original value of the parameter.
–– Decrement; similar to **++**, except the parameter is decremented by 1.
, Separates two arithmetic expressions; the left-hand side is evaluated before the right. The result is the value of the expression on the right-hand side.
= Assignment; variable on the left is set to the value on the right.
***= /= %= += –= <<= >>= &= ^= |=**
Assignment operators; <*var*> <*op*>**=** <*expr*> is the same as <*var*> **=** <*var*> <*op*> **(** <*expr*> **)**.
|| Logical OR; the result is 1 if either argument is non-zero, 0 if not. The right argument is evaluated only if the left argument is zero.
&& Logical AND; the result is 1 if both arguments are non-zero, 0 if not. The right argument is evaluated only if the left argument is non-zero.
| Arithmetic (bit-wise) or.
^ Arithmetic (bit-wise) exclusive-or.
& Arithmetic (bit-wise) and.
== Equal; the result is 1 if both arguments are equal, 0 if not.
!= Not equal; the result is 0 if both arguments are equal, 1 if not.
< Less than; the result is 1 if the left argument is less than the right, 0 if not.
<= >= > Less than or equal, greater than or equal, greater than. See <.
<< >> Shift left (right); the result is the left argument with its bits shifted left (right) by the amount given in the right argument.
+ - * / Addition, subtraction, multiplication, and division.
% Remainder; the result is the remainder of the division of the left argument by the right. The sign of the result is unspecified if either argument is negative.
<*arg1*> **?** <*arg2*> **:** <*arg3*>
If <*arg1*> is non-zero, the result is <*arg2*>, otherwise <*arg3*>.

Co-Processes

A Co-process, which is a pipeline created with the **|&** operator, is an asynchronous process that the shell can both write to (using **print –p**) and read from (using **read –p**). The input and output of the co-process can also be manipulated using **>&p** and **<&p** redirections, repsectively. Once a co-process has been started, another cannot be started until the co-process exits or until the co-process input has been redirected using an **exec** *n***>&p** redirection. If a co-process's input is redirected in this way, the next co-process to be started will share the output with the first co-process, unless the output of the initial co-process has been redirected using an **exec** *n***<&p** redirection.

Some notes concerning co-processes:

- The only way to close the co-process input so that the co-process reads an end-of-file is to redirect the input to a numbered file descriptor and then close that file descriptor. For example:

  ```
  exec 3>&p; exec 3>&-
  ```

  In order for co-processes to share a common output, the shell must keep the write portion of the output pipe open. This means that the end of file will not be detected until all co-processes sharing the co-process output have exited. When they all exit, the shell closes its copy of the pipe. This can be avoided by redirecting the output to a numbered file descriptor (which also causes the shell to close its copy). This behavior is slightly different from the original Korn Shell which closes its copy of the write portion of the co-process when the most-recently-started co-process exits (instead of all sharing co-processes).

- The command **print –p** will ignore SIGPIPE signals during writes if the signal is not being trapped or ignored; the same is not true if the co-process input has been duplicated to another file descriptor and **print –u***n* is used.

Functions

Functions are defined using either KornShell **function** *name* syntax or the Bourne / POSIX shell *name***()** syntax. See below for the difference between the two forms. Functions are like **.**-scripts in that they are executed in the current environment, however, unlike **.**-scripts, shell arguments (i.e., positional parameters, **$1**, etc.) are never visible inside them. When the shell is determining the location of a command, functions are searched after special built-in commands, and before regular and non-regular built-ins, and before the *PATH* is searched.

An existing function may be deleted using **unset –f** *function-name*. A list of functions can be obtained using **typeset +f** and the function definitions can be listed using **typeset –f**. **autoload** (which is an alias for **typeset –fu**) may be used to create undefined functions; when an undefined function is executed, the shell searches the path specified in the *FPATH* parameter for a file with the same name as the function, which, if found is read and executed. If after executing the file, the named function is found to be defined, the function is executed, otherwise, the normal command search is continued (i.e., the shell searches the regular built-in command table and *PATH*). Note that if a command is not found using *PATH*, an attempt is made to autoload a function using *FPATH* (this is an undocumented feature of the original KornShell).

Functions can have two attributes, trace and export, which can be set with **typeset –ft** and **typeset –fx**, respectively. When a traced function is executed, the shell's **xtrace** option is turned on for the functions duration, otherwise the **xtrace** option is turned off.

The export attribute of functions is currently not used. In the original KornShell, exported functions are visible to shell scripts that are executed.

Since functions are executed in the current shell environment, parameter assignments made inside functions are visible after the function completes. If this is not the desired effect, the **typeset** command can be used inside a function to create a local parameter. Note that special parameters (e.g., **$$**, **$!**) can't be scoped in this way.

The exit status of a function is that of the last command executed in the function. A function can be made to finish immediately using the **return** command; this may also be used to explicitly specify the exit status.

Functions defined with the **function** reserved word are be treated differently in the following ways from functions defined with the **()** notation:

- A separate trap / signal environment will be used during the execution of functions. This will mean that traps set inside a function will not affect the shell's traps and signals that are not ignored in the shell (but may be trapped) will have their default effect in a function.
- The EXIT trap, if set in a function, will be executed after the function returns.
- the **$0** parameter is set to the name of the function (Bourne-style functions leave **$0** untouched).
- parameter assignments preceding function calls are not kept in the shell environment (executing Bourne-style functions keep assignments).

POSIX Mode

The shell is intended to be POSIX compliant, however, in some cases, POSIX behaviour is contrary either to the original KornShell behaviour or to user convenience. How the shell behaves in these cases is determined by the state of the posix option (**set +o posix**) — if it is on, the POSIX behaviour is followed, otherwise it is not. The **posix** option is set automatically when the shell starts up if the environment contains the *POSIXLY_CORRECT* parameter.

The following is a list of things that are affected by the state of the **posix** option:

\" inside double quoted **‘..‘** command substitutions:
: In posix mode, the \" is interpreted when the command is interpreted; in non-posix mode, the backslash is stripped before the command substitution is interpreted. For example, **echo "‘echo \"hi\"‘"** produces ‘"hi"’ in posix mode, ‘hi’ in non-posix mode. To avoid problems, use the **$(...)** form of command substitution.

kill –l output:
: In posix mode, signal names are listed on a single line; in non-posix mode, signal numbers, names and descriptions are printed in columns. In future, a new option (**–v** perhaps) will be added to distinguish the two behaviours.

fg exit status:
: In posix mode, the exit status is 0 if no errors occur; in non-posix mode, the exit status is that of the last foregrounded job.

getopts:
: In posix mode, options must start with a **–**; in non-posix mode, options can start with either **–** or **+**.

brace expansion (also known as alternation):

In posix mode, brace expansion is disabled; in non-posix mode, brace expansion enabled. Note that **set –o posix** (or setting the *POSIXLY_CORRECT* parameter) automatically turns the **braceexpand** option off, however it can be explicitly turned on later.

set –: In posix mode, this does not clear the **verbose** or **xtrace** options; in non-posix mode, it does.

set exit status:

In posix mode, the exit status of set is 0 if there are no errors; in non-posix mode, the exit status is that of any command substitutions performed in generating the set command. For example, '**set — 'false'; echo $?**' prints 0 in posix mode, 1 in non-posix mode. This construct is used in most shell scripts that use the old *getopt* command.

argument expansion of **alias**, **export**, **readonly** and **typeset**

In posix mode, normal argument expansion done; in non-posix mode, field splitting, file globbing, brace expansion and (normal) tilde expansion are turned off, and assignment tilde expansion is turned on.

signal specification:

In posix mode, signals can be specified as digits only if signal numbers match POSIX values (i.e., HUP=1, INT=2, QUIT=3, ABRT=6, KILL=9, ALRM=14, and TERM=15); in non-posix mode, signals can be always digits.

alias expansion:

In posix mode, alias expansion is only carried out when reading command words; in non-posix mode, alias expansion is carried out on any word following an alias that ended in a space. For example, the following for loop

```
alias a='for ' i='j'
a i in 1 2; do echo i=$i j=$j; done
```

uses parameter **i** in posix mode, **j** in non-posix mode.

Command Execution

After evaluation of command line arguments, redirections and parameter assignments, the type of command is determined: a special built-in, a function, a regular built-in or the name of a file to execute found using the *PATH* parameter. The checks are made in the above order. Special built-in commands differ from other commands in that the *PATH* parameter is not used to find them, an error during their execution can cause a non-interactive shell to exit and parameter assignments that are specified before the command are kept after the command completes. Just to confuse things, if the posix option is turned off (see **set** command below) some special commands are very special in that no field splitting, file globbing, brace expansion nor tilde expansion is preformed on arguments that look like assignments. Regular built-in commands are different only in that the *PATH* parameter is not used to find them.

The original ksh and POSIX differ somewhat in which commands are considered special or regular:

POSIX special commands

POSIX special commands	**.**	**continue**	**exit**	**return**	**trap**

	:	**eval**	**export**	**set**	**unset**
	break	**exec**	**readonly**	**shift**	
Additional ksh special commands	**builtin**	**times**	**typeset**		
Very special commands (non-posix mode)	**alias**	**readonly**	**set**	**typeset**	
POSIX regular commands	**alias**	**command**	**fg**	**kill**	**umask**
	bg	**false**	**getopts**	**read**	**unalias**
	cd	**fc**	**jobs**	**true**	**wait**
	[	**echo**	**let**	**pwd**	
	print	**test**	**whence**		

In the future, the additional ksh special and regular commands may be treated differently from the POSIX special and regular commands.

Once the type of the command has been determined, any command line parameter assignments are performed and exported for the duration of the command.

The following describes the special and regular built-in commands:

. *file* [*arg1* ...]
: Execute the commands in *file* in the current environment. The file is searched for in the directories of *PATH*. If arguments are given, the positional parameters may be used to access them while *file* is being executed. If no arguments are given, the positional parameters are those of the environment the command is used in.

: [...]
: The null command. Exit status is set to zero.

alias [**–d** | **–t** [**–r**]] [**–x**] [*name1*[**=***value1*] ...]
: Without arguments, **alias** lists all aliases and their values. For any name without a value, its value is listed. Any name with a value defines an alias (see Aliases above).

  The **–x** option sets the export attribute of an alias, or, if no names are given, lists the aliases with the export attribute (exporting an alias currently has no affect).

  The **–t** option indicates that tracked aliases are to be listed / set (values specified on the command line are ignored for tracked aliases). The **–r** option indicates that all tracked aliases are to be reset.

  The **–d** causes directory aliases, which are used in tilde expansion, to be listed or set (see "Tilde Expansion" above).

bg [*job* ...]
: Resume the specified stopped job(s) in the background. If no jobs are specified, **%+** is assumed. This command is only available on systems which support job control. See "Job Control" below for more information.

bind [**–m**] [*key*[**=***editing-command*] ...]
: Set or view the current emacs command editing key bindings / macros. See "Emacs Interactive Input Line Editing" below for a complete description.

break [*level*]
: **break** exits the *level*th innermost **for**, **select**, **until**, or **while** loop. *level* defaults to 1.

builtin *command* [*arg1* ...]
Execute the built-in command *command*.

cd [**–LP**] [*dir*]
Set the working directory to *dir*. If the parameter *CDPATH* is set, it lists the search path for the directory containing *dir*. A null path means the current directory. If *dir* is missing, the home directory **$HOME** is used. If *dir* is **–**, the previous working directory is used (see *OLDPWD* parameter). If **–L** option (logical path) is used or if the **physical** option (see **set** command below) isn't set, references to **..** in *dir* are relative to the path used get to the directory. If **–P** option (physical path) is used or if the **physical** option is set, **..** is relative to the filesystem directory tree. The *PWD* and *OLDPWD* parameters are updated to reflect the current and old wording directory, respectively.

cd [**–LP**] *old new*
The string *new* is substituted for *old* in the current directory, and the shell attempts to change to the new directory.

command [**–pvV**] *cmd* [*arg1* ...]
If neither the **–v** nor **–V** options are given, *cmd* is executed exactly as if the **command** had not been specified, with two exceptions: first, *cmd* cannot be a shell function, and second, special built-in commands lose their specialness (i.e., redirection and utility errors do not cause the shell to exit, and command assignments are not permanent). If the **–p** option is given, a default search path is used instead of the current value of *PATH* (the actual value of the default path is system dependent: on POSIX systems, it is the value returned by **getconf CS_PATH**).

If the **–v** option is given, instead of executing *cmd*, information about what would be executed is given (and the same is done for *arg1* ...): for special and regular built-in commands and functions, their names are simply printed, for aliases, a command that defines them is printed, and for commands found by searching the *PATH* parameter, the full path of the command is printed. If no command is be found, (i.e., the path search fails), nothing is printed and **command** exits with a non-zero status. The **–V** option is like the **–v** option, except it is more verbose.

continue [*levels*]
continue jumps to the beginning of the *level*th innermost **for**, **select**, **until**, or **while** loop. *level* defaults to 1.

echo [**–neE**] [*arg* ...]
Prints its arguments (separated by spaces) followed by a newline, to standard out. The newline is suppressed if any of the arguments contain the backslash sequence **\c**. See **print** command below for a list of other backslash sequences that are recognized.

The options are provided for compatibility with BSD shell scripts: **–n** suppresses the trailing newline, **–e** enables backslash interpretation (a no-op, since this is normally done), and **–E** which suppresses backslash interpretation.

eval *command* ...
The arguments are concatenated (with spaces between them) to form a single string which the shell then parses and executes in the current

environment.

exec [*command* [*arg* ...]]
The command is executed without forking, replacing the shell process.

If no arguments are given, any I/O redirection is permanent and the shell is not replaced. Any file descriptors greater than 2 which are opened or *dup*( )-ed in this way are not made available to other executed commands (i.e., commands that are not built-in to the shell).

exit [*status*]
The shell exits with the specified exit status. If *status* is not specified, the exit status is the current value of the **?** parameter.

export [**–p**] [*parameter*[**=***value*]] ...
Sets the export attribute of the named parameters. Exported parameters are passed in the environment to executed commands. If values are specified, the named parameters also assigned.

If no parameters are specified, the names of all parameters with the export attribute are printed one per line, unless the **–p** option is used, in which case **export** commands defining all exported parameters, including their values, are printed.

false A command that exits with a non-zero status.

fc [**–e** *editor* | **–l** [**–n**]] [**–r**] [*first* [*last*]]
first and **last** select commands from the history. Commands can be selected by history number, or a string specifying the most recent command starting with that string. The **–l** option lists the command on stdout, and **–n** inhibits the default command numbers. The **–r** option reverses the order of the list. Without **–l**, the selected commands are edited by the editor specified with the **–e** option, or if no **–e** is specified, the editor specified by the *FCEDIT* parameter (if this parameter is not set, **$OPENNT_ROOT/bin/ed** is used), and then executed by the shell.

fc [**–e –** | **–s**] [**–g**] [*old***=***new*] [*prefix*]
Re-execute the selected command (the previous command by default) after performing the optional substitution of *old* with *new*. If **–g** is specified, all occurrences of *old* are replaced with *new*. This command is usually accessed with the predefined alias **r='fc –e –'**.

fg [*job* ...]
Resume the specified job(s) in the foreground. If no jobs are specified, **%+** is assumed. This command is only available on systems which support job control. See "Job Control" below for more information.

getopts *optstring name* [*arg* ...]
getopts is used by shell procedures to parse the specified arguments (or positional parameters, if no arguments are given) and to check for legal options. *optstring* contains the option letters that **getopts** is to recognize. If a letter is followed by a colon, the option is expected to have an argument. Arguments containing options must all start with either a **–** or a **+**, options that do not take arguments may be grouped in a single argument. If an option takes an argument and the option character is not the last character of the argument it is found in, the remainder of the argument is taken to be the option's argument, otherwise, the next argument is the option's argument.

Each time **getopts** is invoked, it places the next option in the shell parameter *name* and the index of the next argument to be processed in the shell parameter *OPTIND*. If the option was introduced with a **+**, the option placed in *name* is prefixed with a **+**. When an option requires an argument, **getopts** places it in the shell parameter *OPTARG*. When an illegal option or a missing option argument is encountered a question mark or a colon is placed in *name* (indicating an illegal option or missing argument, respectively) and *OPTARG* is set to the option character that caused the problem. An error message is also printed to standard error if *optstring* does not begin with a colon.

When the end of the options is encountered, **getopts** exits with a non-zero exit status. Options end at the first (non-option argument) argument that does not start with a –, or when a **––** argument is encountered.

Option parsing can be reset by setting *OPTIND* to 1 (this is done automatically whenever the shell or a shell procedure is invoked).

Warning: Changing the value of the shell parameter *OPTIND* to a value other than 1, or parsing different sets of arguments without resetting *OPTIND* may lead to unexpected results.

hash [**–r**] [*name* ...]
Without arguments, any hashed executable command pathnames are listed. The **–r** option causes all hashed commands to be removed from the hash table. Each *name* is searched as if it where a command name and added to the hash table if it is an executable command.

jobs [**–lpn**] [*job* ...]
Display information about the specified jobs; if no jobs are specified, all jobs are displayed. The **–n** option causes information to be displayed only for jobs that have changed state since the last notification. If the **–l** option is used, the process-id of each process in a job is also listed. The **–p** option causes only the process group of each job to be printed. See "Job Control" below for the format of *job* and the displayed job.

kill [**–s** *signame* | *–signum* | *–signame*] { *job* | *pid* | *–pgrp* } ...
Send the specified signal to the specified jobs, process ids, or process groups. If no signal is specified, the signal TERM is sent. If a job is specified, the signal is sent to the job's process group. See "Job Control" below for the format of *job*.

kill –l [*exit-status* ...]
Print the name of the signal that killed a process which exited with the specified *exit-status*es. If no arguments are specified, a list of all the signals, their numbers and a short description of them are printed.

let [*expression* ...]
Each expression is evaluated, see Arithmetic Expressions above. If all expressions are successfully evaluated, the exit status is 0 (1) if the last expression evaluated to non-zero (zero). If an error occurs during the parsing or evaluation of an expression, the exit status is greater than 1. Since expressions may need to be quoted, **((** *expr* **))** is syntactic sugar for **let** "*expr*".

print [**–nprsu***n* | **–R** [**–en**]] [*argument* ...]

print prints its arguments on the standard output, separated by spaces, and terminated with a newline. The **–n** option suppresses the newline. By default, certain C escapes are translated. These include \b, \f, \n, \r, \t, \v, and \0### (# is an octal digit, of which there may be 0 to 3). \c is equivalent to using the **–n** option. \ expansion may be inhibited with the **–r** option. The **–s** option prints to the history file instead of standard output, the **–u** option prints to file descriptor *n* (*n* defaults to 1 if omitted), and the **–p** option prints to the co-process (see Co-Processes above).

The **–R** option is used to emulate, to some degree, the BSD echo command, which does not process \ sequences unless the **–e** option is given. As above, the **–n** option suppresses the trailing newline.

pwd [**–LP**]

Print the present working directory. If **–L** option is used or if the **physical** option (see **set** command below) isn't set, the logical path is printed (i.e., the path used to **cd** to the current directory). If **–P** option (physical path) is used or if the **physical** option is set, the path determined from the filesystem (by following **..** directories to the root directory) is printed.

read [**–prsu***n*] [*parameter* ...]

Reads a line of input from standard input, separate the line into fields using the *IFS* parameter (see Substitution above), and assign each field to the specified parameters. If there are more parameters than fields, the extra parameters are set to null, or alternatively, if there are more fields than parameters, the last parameter is assigned the remaining fields (inclusive of any separating spaces). If no parameters are specified, the *REPLY* parameter is used. If the input line ends in a backslash and the **–r** option was not used, the backslash and newline are stripped and more input is read. If no input is read, **read** exits with a non-zero status.

To display a prompt, append a question mark and the prompt to the first parameter (e.g., **read nfoo?'number of foos: '**).

The **–u***n* and **–p** options cause input to be read from file descriptor *n* or the current co-process, respectively. (See "Co-Processes" for comments on this.) If the **–s** option is used, input is saved to the history file.

readonly [**–p**] [*parameter*[**=***value*]] ...

Sets the readonly attribute of the named parameters. If values are given, parameters are set to them before setting the attribute. Once a parameter is made readonly, it cannot be unset and its value cannot be changed.

If no parameters are specified, the names of all parameters with the readonly attribute are printed one per line, unless the **–p** option is used, in which case **readonly** commands defining all readonly parameters, including their values, are printed.

return [*status*]

Returns from a function or **.** script, with exit status *status*. If no *status* is given, the exit status of the last executed command is used. If used outside of a function or **.** script, it has the same effect as **exit**. Note

that *ksh* treats both profile and **$ENV** files as **.** scripts, while the original KornShell only treats profiles as **.** scripts.

set [±**abCefhkmnpsuvxX**] [±**o** [*option*]] [±**A** *name*] [**—**] [*arg* ...]

The set command can be used to set (**–**) or clear (**+**) shell options, set the positional parameters, or set an array parameter. Options can be changed using the ±**o** *option* syntax, where *option* is the long name of an option, or using the ±*letter* syntax, where *letter* is the option's single letter name (not all options have a single letter name). The following table lists both option letters (if they exist) and long names along with a description of what the option does.

–A		Sets the elements of the array parameter *name* to *arg* ...; If **–A** is used, the array is reset (i.e., emptied) first; if **+A** is used, the first N elements are set (where N is the number of *arg*s), the rest are left untouched.
–a	**allexport**	all new parameters are created with the export attribute
–b	**notify**	Print job notification messages asynchronously, instead of just before the prompt. Only used if job control is enabled (**–m**).
–C	**noclobber**	Prevent **>** redirection from overwriting existing files (**>|** must be used to force an overwrite).
–e	**errexit**	Exit (after executing the *ERR* trap) as soon as an error occurs or a command fails (i.e., exits with a non-zero status). This does not apply to commands whose exit status is explicitly tested by a shell construct such as **if**, **until**, **while**, **&&** or **||** statements.
–f	**noglob**	Do not expand file name patterns.
–h	**trackall**	Create tracked aliases for all executed commands (see Aliases above). On by default for non-interactive shells.
–i	**interactive**	Enable interactive mode — this can only be set / unset when the shell is invoked.
–k	**keyword**	Parameter assignments are recognized anywhere in a command.
–l	**login**	The shell is a login shell — this can only be set / unset when the shell is invoked (see Shell Startup above).
–m	**monitor**	Enable job control (default for interactive shells).
–n	**noexec**	Do not execute any commands — useful for checking the syntax of scripts (ignored if interactive).
–p	**privileged**	Set automatically if, when the shell starts, the read uid or gid does not match the effective uid or gid, respectively. See "Shell Startup" above for a description of what this means.

-r	**restricted**	Enable restricted mode — this option can only be used when the shell is invoked. See "Shell Startup" above for a description of what this means.
–s	**stdin**	If used when the shell is invoked, commands are read from standard input. Set automatically if the shell is invoked with no arguments. When –s is used in the **set** command, it causes the specified arguments to be sorted before assigning them to the positional parameters (or to array *name*, if –A is used).
–u	**nounset**	Referencing of an unset parameter is treated as an error, unless one of the **–**, **+** or **=** modifiers is used.
–v	**verbose**	Write shell input to standard error as it is read.
–x	**xtrace**	Print commands and parameter assignments when they are executed, preceded by the value of *PS4*.
–X	**markdirs**	Mark directories with a trailing **/** during file name generation.
	bgnice	Background jobs are run with lower priority.
	braceexpand	Enable brace expansion (aka, alternation).
	cmd_intitle	Displays the current command line in the terminal window's title bar.
	emacs	Enable BRL emacs-like command line editing (interactive shells only); see Emacs Interactive Input Line Editing.
	gmacs	Enable gmacs-like (Gosling emacs) command line editing (interactive shells only); currently identical to emacs editing except that transpose (^T) acts slightly differently.
	ignoreeof	The shell will not exit on when end-of-file is read, **exit** must be used.
	nohup	Do not kill running jobs with a HUP signal when a login shell exists. Currently set by default, but this will change in the future to be compatible with the original KornShell (which doesn't have this option, but does send the HUP signal).
	nolog	No effect — in the original KornShell, this prevents function definitions from being stored in the history file.
	physical	Causes the **cd** and **pwd** commands to use 'physical' (i.e., the filesystem's) **..** directories instead of 'logical' directories (i.e., the shell handles **..**, which allows the user to be oblivious of symlink links to directories). Clear by default. Note that setting this option does not effect the current value of the *PWD* parameter; only the **cd**

command changes *PWD*. See the **cd** and **pwd** commands above for more details.

posix Enable posix mode. See "POSIX Mode" above.

vi Enable vi-like command line editing (interactive shells only).

viraw No effect — in the original KornShell, unless viraw was set, the vi command line mode would let the tty driver do the work until ESC (^[) was entered. ksh is always in viraw mode.

vi-esccomplete In vi command line editing, do command / file name completion when escape (^[) is entered in command mode.

vi-show8 Prefix characters with the eighth bit set with 'M-'. If this option is not set, characters in the range 128-160 are printed as is, which may cause problems.

vi-tabcomplete In vi command line editing, do command / file name completion when tab (^I) is entered in insert mode.

These options can also be used upon invocation of the shell. The current set of options (with single letter names) can be found in the parameter **–**. **set -o** with no option name will list all the options and whether each is on or off; **set +o** will print the long names of all options that are currently on.

Remaining arguments, if any, are positional parameters and are assigned, in order, to the positional parameters (i.e., **1**, **2**, etc.). If options are ended with **––** and there are no remaining arguments, all positional parameters are cleared. If no options or arguments are given, then the values of all names are printed. For unknown historical reasons, a lone **–** option is treated specially: it clears both the **–x** and **–v** options.

shift [*number*]

The positional parameters *number*+1, *number*+2 etc. are renamed to **1**, **2**, etc. *number* defaults to 1.

test *expression*
[*expression* **]**

test evaluates the *expression* and returns zero status if true, 1 status if false, and greater than 1 if there was an error. It is normally used as the condition command of **if** and **while** statements. The following basic expressions are available:

str *str* has non-zero length. Note that there is the potential for problems if *str* turns out to be an operator (e.g., **-r**) — it is generally better to use a test like **[X**"*str*" **!= X]** instead (double quotes are used in case *str* contains spaces or file globbing characters).

–r *file*	*file* exists and is readable
–w *file*	*file* exists and is writable
–x *file*	*file* exists and is executable
–a *file*	*file* exists (this option is obsolete, and may disappear in future versions)
–e *file*	*file* exists
–f *file*	*file* is a regular file
–d *file*	*file* is a directory
–c *file*	*file* is a character special device
–b *file*	*file* is a block special device
–p *file*	*file* is a named pipe
–u *file*	*file*'s mode has setuid bit set
–g *file*	*file*'s mode has setgid bit set
–k *file*	*file*'s mode has sticky bit set
–s *file*	*file* is not empty
–O *file*	*file*'s owner is the shell's effective user-ID
–G *file*	*file*'s group is the shell's effective group-ID
–S *file*	*file* is a socket
–o *option*	shell *option* is set (see **set** command above for list of options). As a non-standard extension, if the option starts with a **!**, the test is negated; the test always fails if option doesn't exist (thus **[-o** *foo* **-o -o !***foo* **]** returns true if and only if option *foo* exists). The total number of operands to the **[** command determines whether –o is interpreted as the option test or the OR binary operator.
file **–nt** *file*	first *file* is newer than second *file*
file **–ot** *file*	first *file* is older than second *file*
file **–ef** *file*	first *file* is the same file as second *file*
–t *fd*	file descriptor is a tty device. Default value of *fd* is 1.
string	*string* is not empty
–z *string*	*string* is empty
–n *string*	*string* is not empty
string **=** *string*	strings are equal
string **==** *string*	strings are equal
string **!=** *string*	strings are not equal
number **–eq** *number*	numbers compare equal
number **–ne** *number*	numbers compare not equal
number **–ge** *number*	numbers compare greater than or equal
number **–gt** *number*	numbers compare greater than
number **–le** *number*	numbers compare less than or equal
number **–lt** *number*	numbers compare less than

The above basic expressions, in which unary operators have precedence over binary operators, may be combined with the following operators (listed in increasing order of precedence):

expr **–o** *expr*	logical or
expr **–a** *expr*	logical and
! *expr*	logical not
(*expr* **)**	grouping

On operating systems not supporting **/dev/fd/***n* devices (where *n* is a file descriptor number), the **test** command will attempt to fake it for all tests that operate on files (except the **-e** test). I.e., **[-w /dev/fd/2]** tests if file descriptor 2 is writable.

Note that some special rules are applied (courtesy of POSIX) if the number of arguments to **test** or **[** ... **]** is less than five: if leading **!** arguments can be stripped such that only one argument remains then a string length test is performed (again, even if the argument is a unary operator); if leading **!** arguments can be stripped such that three arguments remain and the second argument is a binary operator, then the binary operation is performed (even if first argument is a unary operator, including an unstripped !).

Note: A common mistake is to use **if [$foo = bar]** which fails if parameter **foo** is null or unset, if it has embedded spaces (i.e., *IFS* characters), or if it is a unary operator like **!** or **–n**. Use tests like **if [** "**X$foo**" **= Xbar]** instead.

times Print the accumulated user and system times used by the shell and by processes which have exited that the shell started.

trap [*handler signal* ...]

Sets trap handler that is to be executed when any of the specified signals are received. **Handler** is either a null string, indicating the signals are to be ignored, a minus (**–**), indicating that the default action is to be taken for the signals (see signal(2 or 3)), or a string containing shell commands to be evaluated and executed at the first opportunity (i.e., when the current command completes, or before printing the next *PS1* prompt) after receipt of one of the signals. **Signal** is the name of a signal (e.g., PIPE or ALRM) or the number of the signal (see **kill –l** command above). There are two special signals: *EXIT* (also known as **0**), which is executed when the shell is about to exit, and *ERR* which is executed after an error occurs (an error is something that would cause the shell to exit if the **–e** or **errexit** option were set — see **set** command above). *EXIT* handlers are executed in the environment of the last executed command. Note that for non-interactive shells, the trap handler cannot be changed for signals that were ignored when the shell started.

With no arguments, **trap** lists, as a series of **trap** commands, the current state of the traps that have been set since the shell started.

The original KornShell's **DEBUG** trap and the handling of *ERR* and *EXIT* traps in functions are not yet implemented.

true A command that exits with a zero value.

typeset [[±Ulrtux] [**–L**[*n*]] [**–R**[*n*]] [**–Z**[*n*]] [**–i**[*n*]] | **–f** [**–tux**]] [*name*[**=***value*] ...]

Display or set parameter attributes. With no *name* arguments, parameter attributes are displayed: if no options arg used, the current attributes of all parameters are printed as typeset commands; if an

option is given (or **–** with no option letter) all parameters and their values with the specified attributes are printed; if options are introduced with **+**, parameter values are not printed.

If *name* arguments are given, the attributes of the named parameters are set (**–**) or cleared (**+**). Values for parameters may optionally be specified. If typeset is used inside a function, any newly created parameters are local to the function.

When **–f** is used, typeset operates on the attributes of functions. As with parameters, if no *name*s are given, functions are listed with their values (i.e., definitions) unless options are introduced with **+**, in which case only the function names are reported.

- **–L***n* Left justify attribute: *n* specifies the field width. If *n* is not specified, the current width of a parameter (or the width of its first assigned value) is used. Leading white space (and zeros, if used with the **–Z** option) is stripped. If necessary, values are either truncated or space padded to fit the field width.
- **–R***n* Right justify attribute: *n* specifies the field width. If *n* is not specified, the current width of a parameter (or the width of its first assigned value) is used. Trailing white space are stripped. If necessary, values are either stripped of leading characters or space padded to make them fit the field width.
- **–Z***n* Zero fill attribute: if not combined with **–L**, this is the same as **–R**, except zero padding is used instead of space padding.
- **–i***n* integer attribute: *n* specifies the base to use when displaying the integer (if not specified, the base given in the first assignment is used). Parameters with this attribute may be assigned values containing arithmetic expressions.
- **–U** unsigned integer attribute: integers are printed as unsigned values (only useful when combined with the **–i** option). This option is not in the original KornShell.
- **–f** Function mode: display or set functions and their attributes, instead of parameters.
- **–l** Lower case attribute: all upper case characters in values are converted to lower case. (In the original KornShell, this parameter meant 'long integer' when used with the `-i` option).
- **–r** Readonly attribute: parameters with the this attribute may not be assigned to or unset. Once this attribute is set, it can not be turned off.
- **–t** Tag attribute: has no meaning to the shell; provided for application use. For functions, **–t** is the trace attribute. When functions with the trace attribute are executed, the **xtrace** (**–x**) shell option is temporarily turned on.
- **–u** Upper case attribute: all lower case characters in values are converted to upper case. (In the original KornShell, this parameter meant 'unsigned integer' when used with the `-i` option, which meant upper case letters would never be used for bases greater than 10. See the `-U` option). For functions, `-u` is the undefined attribute. See "Functions" above for

the implications of this.

–x Export attribute: parameters (or functions) are placed in the environment of any executed commands. Exported functions are not implemented yet.

ulimit [**–acdfHlmnpsStvw**] [*value*]
Has no effect.

umask [**–S**] [*mask*]
Display or set the file permission creation mask, or umask (see *umask*( )). If the **–S** option is used, the mask displayed or set is symbolic, otherwise it is an octal number.

Symbolic masks are like those used by *chmod*: [**ugoa**]{{**=+-**}{**rwx**}*}+[**,**...] in which the first group of characters is the *who* part, the second group is the *op* part, and the last group is the *perm* part. The *who* part specifies which part of the umask is to be modified. The letters mean:

u the user permissions

g the group permissions

o the other permissions (non-user, non-group)

a all permissions (user, group and other)

The *op* part indicates how the *who* permissions are to be modified:

= set

+ added to

– removed from

The *perm* part specifies which permissions are to be set, added or removed:

r read permission

w write permission

x execute permission

When symbolic masks are used, they describe what permissions may be made available (as opposed to octal masks in which a set bit means the corresponding bit is to be cleared). Example: 'ug=rwx,o=' sets the mask so files will not be readable, writable or executable by 'others', and is equivalent (on most systems) to the octal mask '07'.

unalias [**–adt**] [*name1* ...]
The aliases for the given names are removed. If the **–a** option is used, all aliases are removed. If the **–t** or **–d** options are used, the indicated operations are carried out on tracked or directory aliases, respectively.

unset [**–fv**] *parameter* ...
Unset the named parameters (**–v**, the default) or functions (**–f**). The exit status is non-zero if any of the parameters were already unset, zero otherwise.

wait [*job*]
Wait for the specified job(s) to finish. The exit status of wait is that of

the last specified job: if the last job is killed by a signal, the exit status is 128 + the number of the signal (see **kill –l** *exit-status* above); if the last specified job can't be found (because it never existed, or had already finished), the exit status of wait is 127. See "Job Control" below for the format of *job*. **Wait** will return if a signal for which a trap has been set is received, or if a HUP, INT or QUIT signal is received.

If no jobs are specified, **wait** waits for all currently running jobs (if any) to finish and exits with a zero status. If job monitoring is enabled, the completion status of jobs is printed (this is not the case when jobs are explicitly specified).

whence [**–pv**] [name ...]

For each name, the type of command is listed (reserved word, built-in, alias, function, tracked alias or executable). If the **–p** option is used, a path search done even if *name* is a reserved word, alias, etc. Without the **–v** option, **whence** is similar to **command –v** except that **whence** will find reserved words and won't print aliases as alias commands; with the **–v** option, **whence** is the same as **command –V**. Note that for **whence**, the **–p** option does not affect the search path used, as it does for **command**. If the type of one or more of the names could not be determined, the exit status is non-zero.

Job Control

Job control refers to the shell's ability to monitor and control **jobs**, which are processes or groups of processes created for commands or pipelines. At a minimum, the shell keeps track of the status of the background (i.e., asynchronous) jobs that currently exist; this information can be displayed using the **jobs** command. If job control is fully enabled (using **set –m** or **set –o monitor**), as it is for interactive shells, the processes of a job are placed in their own process group, foreground jobs can be stopped by typing the suspend character from the terminal (normally ^Z), jobs can be restarted in either the foreground or background, using the **fg** and **bg** commands, respectively, and the state of the terminal is saved or restored when a foreground job is stopped or restarted, respectively.

Note that only commands that create processes (e.g., asynchronous commands, subshell commands, and non-built-in, non-function commands) can be stopped; commands like **read** cannot be.

When a job is created, it is assigned a job-number. For interactive shells, this number is printed inside **[..]**, followed by the process-ids of the processes in the job when an asynchronous command is run. A job may be referred to in **bg**, **fg**, **jobs**, **kill** and **wait** commands either by the process id of the last process in the command pipeline (as stored in the **$!** parameter) or by prefixing the job-number with a percent sign (**%**). Other percent sequences can also be used to refer to jobs:

The most recently stopped job, or, if there are no stopped jobs, the oldest

%+	running job.
%%, **%**	Same as **%+**.
%–	The job that would be the **%+** job, if the later did not exist.
%*n*	The job with job-number *n*.
%?*string*	The job containing the string *string* (an error occurs if multiple jobs are matched).
%*string*	The job starting with string *string* (an error occurs if multiple jobs are matched).

When a job changes state (e.g., a background job finishes or foreground job is stopped), the shell prints the following status information: **[***number***]** *flag status command* where

number Is the job-number of the job.

flag Is **+** or **-** if the job is the **%+** or **%-** job, respectively, or space if it is neither.

status Indicates the current state of the job and can be

Running
The job has neither stopped or exited (note that running does not necessarily mean consuming CPU time — the process could be blocked waiting for some event).

Done [**(***number***)**]
The job exited. *number* is the exit status of the job, which is omitted if the status is zero.

Stopped [**(***signal***)**]
The job was stopped by the indicated *signal* (if no signal is given, the job was stopped by SIGTSTP).

signal-description [**(core dumped)**]
The job was killed by a signal (e.g., Memory fault, Hangup, etc. — use **kill –l** for a list of signal descriptions). The **(core dumped)** message indicates the process created a core file.

command
Is the command that created the process. If there are multiple processes in the job, then each process will have a line showing its *command* and possibly its *status*, if it is different from the status of the previous process.

When an attempt is made to exit the shell while there are jobs in the stopped state, the shell warns the user that there are stopped jobs and does not exit. If another attempt is immediately made to exit the shell, the stopped jobs are sent a HUP signal and the shell exits. Similarly, if the **nohup** option is not set and there are running jobs when an attempt is made to exit a login shell, the shell warns the user and does not exit. If another attempt is immediately made to exit the shell, the running jobs are sent a HUP signal and the shell exits.

Emacs Interactive Input Line Editing

When the **emacs** option is set, interactive input line editing is enabled. **Warning**: This mode is slightly different from the emacs mode in the original KornShell and the 8th bit is stripped in emacs mode. In this mode various editing commands (typically bound to one or more control characters) cause immediate actions without waiting for a new-line. Several editing commands are bound to particular control characters when the shell is invoked; these bindings can be changed using the following commands:

bind The current bindings are listed.

bind *string*=[*editing-command*]
The specified editing command is bound to the given **string**, which should consist of a control character (which may be written using caret notation ^*X*), optionally preceded by one of the two prefix characters. Future input of the *string* will cause the editing command to be immediately invoked. Note that although only two prefix characters (usually ESC and ^X) are supported, some multi-character sequences can be supported. The following binds the arrow keys on an ANSI terminal, or xterm (these are in the default bindings). Of course some escape sequences won't work out quite this nicely:

```
bind '^[['=prefix-2
bind '^XA'=up-history
bind '^XB'=down-history
bind '^XC'=forward-char
bind '^XD'=backward-char
```

bind –l Lists the names of the functions to which keys may be bound.

bind –m *string*=[*substitute*]
The specified input *string* will afterwards be immediately replaced by the given *substitute* string, which may contain editing commands.

The following is a list of editing commands available. Each description starts with the name of the command, a *n*, if the command can be prefixed with a count, and any keys the command is bound to by default (written using caret notation, e.g., ASCII ESC character is written as ^[). A count prefix for a command is entered using the sequence **^[***n*, where *n* is a sequence of 1 or more digits; unless otherwise specified, if a count is omitted, it defaults to 1. Note that editing command names are used only with the **bind** command. Furthermore, many editing commands are useful only on terminals with a visible cursor. The default bindings were chosen to resemble corresponding EMACS key bindings. The users tty characters (e.g., ERASE) are bound to reasonable substitutes and override the default bindings.

abort ^G
Useful as a response to a request for a **search-history** pattern in order to abort the search.

auto-insert *n*
Simply causes the character to appear as literal input. Most ordinary characters are bound to this.

backward-char
Moves the cursor backward *n* characters.

backward-word *n* **^[B**
Moves the cursor backward to the beginning of a word; words consist of alphanumerics, underscore (_) and dollar ($).

beginning-of-history ^[<
Moves to the beginning of the history.

beginning-of-line ^A
Moves the cursor to the beginning of the edited input line.

capitalize-word *n* **^[c**, **^[C**
Uppercase the first character in the next *n* words, leaving the cursor past the end of the last word.

comment ^[#
If the current line does not begin with a comment character, one is added at the beginning of the line and the line is entered (as if return had been pressed), otherwise the existing comment characters are removed and the cursor is placed at the beginning of the line.

complete ^[^[
Automatically completes as much as is unique of the command name or the file name containing the cursor. If the entire remaining command or file name is unique a space is printed after its completion, unless it is a directory name in which case **/** is appended. If there is no command or file name with the current partial word as its prefix, a bell character is output (usually causing a audio beep).

complete-command ^X^[
Automatically completes as much as is unique of the command name having the partial word up to the cursor as its prefix, as in the **complete** command described above.

complete-file ^[^X
Automatically completes as much as is unique of the file name having the partial word up to the cursor as its prefix, as in the **complete** command described above.

complete-list ^[=
List the possible completions for the current word.

delete-char-backward *n ERASE*, **^?**, **^H**
Deletes *n* characters before the cursor.

delete-char-forward *n*
Deletes *n* characters after the cursor.

delete-word-backward *n* **^[ERASE**, **^[^?**, **^[^H**, **^[h**
Deletes *n* words before the cursor.

delete-word-forward *n* **^[d**
Deletes characters after the cursor up to the end of *n* words.

down-history *n* **^N**
Scrolls the history buffer forward *n* lines (later). Each input line originally starts just after the last entry in the history buffer, so **down-history** is not useful until either **search-history** or **up-history** has been performed.

downcase-word *n* **^[L**, **^[l**
Lowercases the next *n* words.

end-of-history ^[>
Moves to the end of the history.

end-of-line ^E
Moves the cursor to the end of the input line.

eot ^_ Acts as an end-of-file; this is useful because edit-mode input disables normal terminal input canonicalization.

eot-or-delete *n* **^D**
Acts as eot if alone on a line; otherwise acts as delete-char-forward.

error Error (ring the bell).

exchange-point-and-mark ^X^X
Places the cursor where the mark is, and sets the mark to where the cursor was.

expand-file ^[*
Appends a * to the current word and replaces the word with the result of performing file globbing on the word. If no files match the pattern, the bell is rung.

forward-char *n* **^F**
Moves the cursor forward *n* characters.

forward-word *n* **^[f**
Moves the cursor forward to the end of the *n*th word.

goto-history *n* **^[g**
Goes to history number *n*.

kill-line KILL
Deletes the entire input line.

kill-region ^W
Deletes the input between the cursor and the mark.

kill-to-eol *n* **^K**
Deletes the input from the cursor to the end of the line if *n* is not specified, otherwise deletes characters between the cursor and column *n*.

list ^[? Prints a sorted, columnated list of command names or file names (if any) that can complete the partial word containing the cursor. Directory names have **/** appended to them.

list-command ^X?
Prints a sorted, columnated list of command names (if any) that can complete the partial word containing the cursor.

list-file ^X^Y
Prints a sorted, columnated list of file names (if any) that can complete the partial word containing the cursor. File type indicators are appended as described under **list** above.

newline ^J, **^M**
Causes the current input line to be processed by the shell. The current cursor position may be anywhere on the line.

newline-and-next ^O
Causes the current input line to be processed by the shell, and the next line from history becomes the current line. This is only useful after an up-history or search-history.

no-op QUIT
This does nothing.

prefix-1 ^[
Introduces a 2-character command sequence.

prefix-2 ^X

prefix-2 ^[[
Introduces a 2-character command sequence.

prev-hist-word *n* **^[.**, **^[_**

The last (*n*th) word of the previous command is inserted at the cursor.

quote ^^ The following character is taken literally rather than as an editing command.

redraw ^L
Reprints the prompt string and the current input line.

search-character-backward *n* **^[^]**
Search backward in the current line for the *n*th occurence of the next character typed.

search-character-forward *n* **^]**
Search forward in the current line for the *n*th occurence of the next character typed.

search-history ^R
Enter incremental search mode. The internal history list is searched backwards for commands matching the input. An initial ^ in the search string anchors the search. The abort key will leave search mode. Other commands will be executed after leaving search mode. Successive **search-history** commands continue searching backward to the next previous occurrence of the pattern. The history buffer retains only a finite number of lines; the oldest are discarded as necessary.

set-mark-command ^[<space>
Set the mark at the cursor position.

stuff On systems supporting it, pushes the bound character back onto the terminal input where it may receive special processing by the terminal handler. This is useful for the BRL **^T** mini-systat feature, for example.

stuff-reset
Acts like **stuff**, then aborts input the same as an interrupt.

transpose-chars ^T
If at the end of line, or if the **gmacs** option is set, this exchanges the two previous characters; otherwise, it exchanges the previous and current characters and moves the cursor one character to the right.

up-history *n* **^P**
Scrolls the history buffer backward *n* lines (earlier).

upcase-word *n* **^[U**, **^[u**
Uppercases the next *n* words.

version ^V
Display the version of ksh. The current edit buffer is restored as soon as any key is pressed (the key is then processed, unless it is a space).

yank ^Y Inserts the most recently killed text string at the current cursor position.

yank-pop ^[y
Immediately after a **yank**, replaces the inserted text string with the next previous killed text string.

Vi Interactive Input Line Editing

The vi command line editor in ksh has basically the same commands as the vi editor (see *vi*), with the following exceptions:

- you start out in insert mode,
- there are file name and command completion commands (**=**, ****, *****, **^X**, **^E**, **^F** and, optionally, <tab>),

- the **_** command is different (in ksh it is the last argument command, in vi it goes to the start of the current line),
- the **/** and **G** commands move in the opposite direction as the **j** command
- and commands which don't make sense in a single line editor are not available (e.g., screen movement commands, ex **:** commands, etc.).

Note that the **^X** stands for control-X; also <esc>, <space> and <tab> are used for escape, space and tab, respectively (no kidding).

Like vi, there are two modes: insert mode and command mode. In insert mode, most characters are simply put in the buffer at the current cursor position as they are typed, however, some characters are treated specially. In particular, the following characters are taken from current tty settings (see *stty*) and have their usual meaning (normal values are in parentheses): kill (**^U**), erase (**^?**), werase (**^W**), eof (**^D**), intr (**^C**) and quit (^\). In addition to the above, the following characters are also treated specially in insert mode:

^H	erases previous character
^V	literal next: the next character typed is not treated specially (can be used to insert the characters being described here)
^J ^M	end of line: the current line is read, parsed and executed by the shell
<esc>	puts the editor in command mode (see below)
^E	command and file name enumeration (see below)
^F	command and file name completion (see below). If used twice in a row, the list of possible completions is displayed; if used a third time, the completion is undone.
^X	command and file name expansion (see below)
<tab>	optional file name and command completion (see **^F** above), enabled with **set –o vi-tabcomplete**

If a line is longer that the screen width (see *COLUMNS* parameter), a **>**, **+** or **<** character is displayed in the last column indicating that there are more characters after, before and after, or before the current position, respectively. The line is scrolled horizontally as necessary.

In command mode, each character is interpreted as a command. Characters that don't correspond to commands, are illegal combinations of commands or are commands that can't be carried out all cause beeps. In the following command descriptions, a *n* indicates the command may be prefixed by a number (e.g., **10l** moves right 10 characters); if no number prefix is used, *n* is assumed to be 1 unless otherwise specified. The term 'current position' refers to the position between the cursor and the character preceding the cursor. A 'word' is a sequence of letters, digits and underscore characters or a sequence of non-letter, non-digit, non-underscore, non-white-space characters (e.g., ab2*&^ contains two words) and a 'big-word' is a sequence of non-white-space characters.

Special ksh vi commands

The following commands are not in, or are different from, the normal vi file editor:

*n***_** Insert a space followed by the *n*th big-word from the last command in the history at the current position and enter insert mode; if *n* is not specified, the last word is inserted.

Insert the comment character (**#**) at the start of the current line and return the line to the shell (equivalent to **I#^J**).

*n***g** Like **G**, except if *n* is not specified, it goes to the most recent remembered line.

*n***v** Edit line *n* using the vi editor; if *n* is not specified, the current line is edited. The actual command executed is '**fc –e ${VISUAL:-${EDITOR:-vi}}** *n*'.

***** and **^X**
Command or file name expansion is applied to the current big-word (with an appended *, if the word contains no file globbing characters) - the big-word is replaced with the resulting words. If the current big-word is the first on the line (or follows one of the following characters: **;**, **|**, **&**, **(**, **)**) and does not contain a slash (**/**) then command expansion is done, otherwise file name expansion is done. Command expansion will match the big-word against all aliases, functions and built-in commands as well as any executable files found by searching the directories in the *PATH* parameter. File name expansion matches the big-word against the files in the current directory. After expansion, the cursor is placed just past the last word and the editor is in insert mode.

*n*****, *n***^F**, *n***<tab>**, and *n***<exc>**
command / file name completion: replace the current big-word with the longest unique match obtained after performing command / file name expansion. **<tab>** is only recognized if the **vi-tabcomplete** option is set, while **<esc>** is only recognized if the **vi-esccomplete** option is set (see **set –o**), If *n* is specified, the *n*th possible completion is selected (as reported by the command / file name enumeration command).

= and **^E**
command / file name enumeration: list all the commands or files that match the current big-word.

^V display the version of *ksh*; it is displayed until another key is pressed (this key is ignored).

@*c* macro expansion: execute the commands found in the alias *_c*.

Intra-line movement commands

The following commands control movement within the line in vi-edit mode.

*n***h** and *n***^H**
Move left *n* characters.

*n***l** and *n***<space>**
Move right *n* characters.

0 Move to column 0.

^ Move to the first non white-space character.

n **|** Move to column *n*.
$ Move to the last character.
*n***b** Move back *n* words.
*n***B** Move back *n* big-words.
*n***e** Move forward to the end the word, *n* times.
*n***E** Move forward to the end the big-word, *n* times.
*n***w** Move forward *n* words.
*n***W** Move forward *n* big-words.
% Find match: the editor looks forward for the nearest parenthesis, bracket or brace and then moves the to the matching parenthesis, bracket or brace.
*n***f***c* Move forward to the *n*th occurrence of the character *c*.
*n***F***c* Move backward to the *n*th occurrence of the character *c*.
*n***t***c* Move forward to just before the *n*th occurrence of the character *c*.
*n***T***c* Move backward to just before the *n*th occurrence of the character *c*.
*n***;** Repeats the last **f**, **F**, **t** or **T** command.
*n***,** Repeats the last **f**, **F**, **t** or **T** command, but moves in the opposite direction.

Inter-line movement commands

The following commands control movement between command lines in vi-edit mode.

*n***j** and *n***+** and *n***^N**
Move to the *n*th next line in the history.
*n***k** and *n***-** and *n***^P**
Move to the *n*th previous line in the history.
*n***G** Move to line *n* in the history; if *n* is not specified, the number first remembered line is used.
*n***g** Like **G**, except if *n* is not specified, it goes to the most recent remembered line.
n/string Search backward through the history for the *n*th line containing *string*; if *string* starts with **^**, the remainder of the string must appear at the start of the history line for it to match.
*n***?***string* Same as **/**, except it searches forward through the history.
*n***n** Search for the *n*th occurrence of the last search string; the direction of the search is the same as the last search.
*n***N** Search for the *n*th occurrence of the last search string; the direction of the search is the opposite of the last search.

Edit commands

The following commands edit the command line.

*n***a** Append text *n* times: goes into insert mode just after the current position. The append is only replicated if command mode is re-entered (i.e., <esc> is used).
*n***A** Same as **a**, except it appends at the end of the line.
*n***i** Insert text *n* times: goes into insert mode at the current position. The insertion is only replicated if command mode is re-entered (i.e., <esc> is used).
*n***I** Same as **i**, except the insertion is done just before the first non-blank

character.

*n***s** Substitute the next *n* characters (i.e., delete the characters and go into insert mode).

S Substitute whole line: all characters from the first non-blank character to the end of line are deleted and insert mode is entered.

*n***c***move-cmd*
Change from the current position to the position resulting from *n* *move-cmd*s (i.e., delete the indicated region and go into insert mode); if *move-cmd* is **c**, the line starting from the first non-blank character is changed.

C Change from the current position to the end of the line (i.e., delete to the end of the line and go into insert mode).

*n***x** Delete the next *n* characters.

*n***X** Delete the previous *n* characters.

D Delete to the end of the line.

*n***d***move-cmd*
Delete from the current position to the position resulting from *n* *move-cmd*s; *move-cmd* is a movement command (see above) or **d**, in which case the current line is deleted.

*n***r***c* Replace the next *n* characters with the character *c*.

*n***R** Replace: enter insert mode but overwrite existing characters instead of inserting before existing characters. The replacement is repeated *n* times.

*n***~** Change the case of the next *n* characters.

*n***y***move-cmd*
Yank from the current position to the position resulting from *n* *move-cmd*s into the yank buffer; if *move-cmd* is **y**, the whole line is yanked.

Y Yank from the current position to the end of the line.

*n***p** Paste the contents of the yank buffer just after the current position, *n* times.

*n***P** Same as **p**, except the buffer is pasted at the current position.

Miscellaneous vi commands

^J and **^M**
The current line is read, parsed and executed by the shell.

^L and **^R**
Redraw the current line.

*n***.** Redo the last edit command *n* times.

u Undo the last edit command.

U Undo all changes that have been made to the current line.

intr and *quit*
The interrupt and quit terminal characters cause the current line to be deleted and a new prompt to be printed.

FILES

~/.profile
Personal startup file.

$OPENNT_ROOT/etc/profile
System-wide startup file; the **etc** directory may have a different root, depending upon where the INTERIX distribution is installed.

AUTHORS

This shell is based on the public domain 7th edition Bourne shell clone by Charles Forsyth and parts of the BRL shell by Doug A. Gwyn, Doug Kingston, Ron Natalie, Arnold Robbins, Lou Salkind and others. The first release was created by Eric Gisin, and it was subsequently maintained by John R. MacMillan (chance!john@sq.sq.com), and Simon J. Gerraty (sjg@zen.void.oz.au). The current maintainer is Michael Rendell (michael@cs.mun.ca). The CONTRIBUTORS file in the source distribution contains a more complete list of people and their part in the shell's development.

SEE ALSO

awk, *sh*, *csh*, *ed*, *getconf*, *sed*, *stty*, *vi*

dup(), *exec*(), *getgid*(), *getuid*(), *open*(), *pipe*(), *wait*(), *getopt*(), *rand*(), *signal*(), *system*()

The KornShell Command and Programming Language, Morris Bolsky and David Korn, 1989, ISBN 0-13-516972-0.

UNIX Shell Programming, Stephen G. Kochan, Patrick H. Wood, Hayden.

IEEE Standard for information Technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities, IEEE Inc, 1993, ISBN 1-55937-255-9.

NAME

ld – the GNU linker

SYNOPSIS

```
ld [-EgiMSstVvXx] [-o output] objfile ...
   [-A architecture] [-b input-format]
   [-Bstatic] [-Bdynamic] [-Bsymbolic]
   [-c commandfile] [--cref] [-d|-dc|-dp]
   [-defsym symbol=expression] [-e entry]
   [-embedded-relocs] [-export-dynamic] [-f name]
   [--auxiliary name] [-F name] [--filter name]
   [-format input-format] [-G size] [-h name]
   [-soname name] [--help] [-l ar] [-L searchdir]
   [-Map mapfile] [-m emulation] [-n|-N]
   [-noinhibit-exec] [-no-keep-memory] [-oformat output-format]
   [-R filename] [-relax] [-r|-Ur]
   [-rpath directory] [-rpath-link directory]
   [-shared] [-sort-common] [-split-by-reloc count]
   [-split-by-file] [-T commandfile]  [-Ttext textorg]
   [-Tdata dataorg] [-Tbss bssorg] [-u sym]
   [--verbose] [--version] [-warn-common] [-warn-constructors]
   [-warn-multiple-gp] [-warn-once] [-warn-section-align]
   [--whole-archive] [--no-whole-archive] [--wrap symbol]
```

DESCRIPTION

ld combines a number of object and archive files, relocates their data and ties up symbol references. Often the last step in building a new compiled program to run is a call to *ld*.

ld accepts Linker Command Language files to provide explicit and total control over the linking process. This man page does not describe the command language; see the *ld* entry in **info**, or the manual *ld: the GNU linker*, for full details on the command language and on other aspects of the GNU linker.

This version of *ld* uses the general purpose BFD libraries to operate on object files. This allows *ld* to read, combine, and write object files in many different formats—for example, COFF or **a.out**. Different formats may be linked together to produce any available kind of object file. You can use **objdump –i** to get a list of formats supported on various architectures; see *objdump*.

Aside from its flexibility, the GNU linker is more helpful than other linkers in providing diagnostic information. Many linkers abandon execution immediately upon encountering an error; whenever possible, *ld* continues executing, allowing you to identify other errors (or, in some cases, to get an output file in spite of the error).

The GNU linker *ld* is meant to cover a broad range of situations, and to be as compatible as possible with other linkers. As a result, you have many choices to control its behavior through the command line, and through environment variables.

OPTIONS

The command-line options to *ld* may be specified in any order, and may be repeated at will. For the most part, repeating an option with a different argument will either have no further effect, or override prior occurrences (those further to the left on the command line) of an option.

The exceptions—which may meaningfully be used more than once—are **–A**, **–b** (or its synonym **–format**), **–defsym**, **–L**, **–l**, **–R**, and **–u**.

The list of object files to be linked together, shown as *objfile*, may follow, precede, or be mixed in with command-line options; save that an *objfile* argument may not be placed between an option flag and its argument.

Usually the linker is invoked with at least one object file, but other forms of binary input files can also be specified with **–l**, **–R**, and the script command language. If *no* binary input files at all are specified, the linker does not produce any output, and issues the message "No input files".

Option arguments must either follow the option letter without intervening whitespace, or be given as separate arguments immediately following the option that requires them.

–A *architecture*

In the current release of *ld*, this option is useful only for the Intel 960 family of architectures. In that *ld* configuration, the *architecture* argument is one of the two-letter names identifying members of the 960 family; the option specifies the desired output target, and warns of any incompatible instructions in the input files. It also modifies the linker's search strategy for archive libraries, to support the use of libraries specific to each particular architecture, by including in the search loop names suffixed with the string identifying the architecture.

For example, if your *ld* command line included **–ACA** as well as **–ltry**, the linker would look (in its built-in search paths, and in any paths you specify with **–L**) for a library with the names

```
try
libtry.a
tryca
libtryca.a
```

The first two possibilities would be considered in any event; the last two are due to the use of **–ACA**.

Future releases of *ld* may support similar functionality for other architecture families.

You can meaningfully use **–A** more than once on a command line, if an architecture family allows combination of target architectures; each use will add another pair of name variants to search for when **–l** specifies a library.

–b *input-format*
Specify the binary format for input object files that follow this option on the command line. You don't usually need to specify this, as *ld* is configured to expect as a default input format the most usual format on each machine. *input-format* is a text string, the name of a particular format supported by the BFD libraries. **–format** *input-format* has the same effect, as does the script command **TARGET**.

You may want to use this option if you are linking files with an unusual binary format. You can also use **–b** to switch formats explicitly (when linking object files of different formats), by including **–b** *input-format* before each group of object files in a particular format.

The default format is taken from the environment variable *GNUTARGET*. You can also define the input format from a script, using the command **TARGET**.

–Bstatic Do not link against shared libraries. This is only meaningful on platforms for which shared libraries are supported.

–Bdynamic
Link against dynamic libraries. This is only meaningful on platforms for which shared libraries are supported. This option is normally the default on such platforms.

–Bsymbolic
When creating a shared library, bind references to global symbols to the definition within the shared library, if any. Normally, it is possible for a program linked against a shared library to override the definition within the shared library. This option is only meaningful on ELF platforms which support shared libraries.

–c *commandfile*
Directs *ld* to read link commands from the file *commandfile*. These commands will completely override *ld*'s default link format (rather than adding to it); *commandfile* must specify everything necessary to describe the target format.

You may also include a script of link commands directly in the command line by bracketing it between { and } characters.

––cref Output a cross reference table. If a linker map file is being generated, the cross reference table is printed to the map file. Otherwise, it is printed on the standard output.

–d, **–dc**, **–dp**
These three options are equivalent; multiple forms are supported for compatibility with other linkers. Use any of them to make *ld* assign space to common symbols even if a relocatable output file is specified (**–r**). The script command **FORCE_COMMON_ALLOCATION** has the same effect.

–defsym *symbol-expression*
Create a global symbol in the output file, containing the absolute address given by *expression*. You may use this option as many times as necessary to define multiple symbols in the command line. A

limited form of arithmetic is supported for the *expression* in this context: you may give a hexadecimal constant or the name of an existing symbol, or use **+** and **–** to add or subtract hexadecimal constants or symbols. If you need more elaborate expressions, consider using the linker command language from a script.

–e *entry*
Use *entry* as the explicit symbol for beginning execution of your program, rather than the default entry point. for a discussion of defaults and other ways of specifying the entry point.

–embedded–relocs
This option is only meaningful when linking MIPS embedded PIC code, generated by the **–membedded–pic** option to the GNU compiler and assembler. It causes the linker to create a table which may be used at runtime to relocate any data which was statically initialized to pointer values. See the code in **testsuite/ld-empic** for details.

–E, **–export–dynamic**
When creating an ELF file, add all symbols to the dynamic symbol table. Normally, the dynamic symbol table contains only symbols which are used by a dynamic object. This option is needed for some uses of *dlopen.*( )

–f *name*

––auxiliary *name*
When creating an ELF shared object, set the internal DT_AUXILIARY field to the specified name. This tells the dynamic linker that the symbol table of the shared object should be used as an auxiliary filter on the symbol table of the shared object *name.*

–F *name*

––filter *name*
When creating an ELF shared object, set the internal DT_FILTER field to the specified name. This tells the dynamic linker that the symbol table of the shared object should be used as a filter on the symbol table of the shared object *name.*

–format *input–format*
Synonym for **–b** *input–format*.

–g Accepted, but ignored; provided for compatibility with other tools.

–G *size*
Set the maximum size of objects to be optimized using the GP register to *size* under MIPS ECOFF. Ignored for other object file formats.

–h *name*

–soname *name*
When creating an ELF shared object, set the internal DT_SONAME field to the specified name. When an executable is linked with a shared object which has a DT_SONAME field, then when the executable is run the dynamic linker will attempt to load the shared object specified by the DT_SONAME field rather than the using the

file name given to the linker.

––help Print a summary of the command-line options on the standard output and exit. This option and **—version** begin with two dashes instead of one for compatibility with other GNU programs. The other options start with only one dash for compatibility with other linkers.

–i Perform an incremental link (same as option **–r**).

–l *ar* Add an archive file *ar* to the list of files to link. This option may be used any number of times. *ld* will search its path-list for occurrences of **lib***ar***.a** for every *ar* specified.

–L *searchdir*
This command adds path *searchdir* to the list of paths that *ld* will search for archive libraries. You may use this option any number of times.

The default set of paths searched (without being specified with **–L**) depends on what emulation mode *ld* is using, and in some cases also on how it was configured. The paths can also be specified in a link script with the **SEARCH_DIR** command.

–M Print (to the standard output file) a link map—diagnostic information about where symbols are mapped by *ld*, and information on global common storage allocation.

–Map *mapfile*
Print to the file *mapfile* a link map—diagnostic information about where symbols are mapped by *ld*, and information on global common storage allocation.

–m *emulation*
Emulate the *emulation* linker. You can list the available emulations with the **—verbose** or **–V** options. This option overrides the compiled-in default, which is the system for which you configured *ld*.

–N Specifies readable and writable **text** and **data** sections. If the output format supports traditional magic numbers, the output is marked as **OMAGIC**.

When you use the **–N** option, the linker does not page-align the data segment.

–n Sets the text segment to be read only, and **NMAGIC** is written if possible.

–noinhibit–exec
Normally, the linker will not produce an output file if it encounters errors during the link process. With this flag, you can specify that you wish the output file retained even after non-fatal errors.

–no–keep–memory
The linker normally optimizes for speed over memory usage by caching the symbol tables of input files in memory. This option tells the linker to instead optimize for memory usage, by rereading the symbol tables as necessary. This may be required if the linker runs out of memory space while linking a large executable.

–o *output*

output is a name for the program produced by *ld*; if this option is not specified, the name **a.out** is used by default. The script command **OUTPUT** can also specify the output file name.

–oformat *output–format*

Specify the binary format for the output object file. You don't usually need to specify this, as *ld* is configured to produce as a default output format the most usual format on each machine. *output-format* is a text string, the name of a particular format supported by the BFD libraries. The script command **OUTPUT_FORMAT** can also specify the output format, but this option overrides it.

–R *filename*

Read symbol names and their addresses from *filename*, but do not relocate it or include it in the output. This allows your output file to refer symbolically to absolute locations of memory defined in other programs.

–r Generates relocatable output—i.e., generate an output file that can in turn serve as input to *ld*. This is often called *partial* linking. As a side effect, in environments that support traditional magic numbers, this option also sets the output file's magic number to **OMAGIC**. If this option is not specified, an absolute file is produced. When linking C++ programs, this option *will not* resolve references to constructors; **–Ur** is an alternative.

This option does the same as **–i**.

–relax An option with machine dependent effects. Currently this option is only supported on the H8/300.

On some platforms, use this option to perform global optimizations that become possible when the linker resolves addressing in your program, such as relaxing address modes and synthesizing new instructions in the output object file.

On platforms where this is not supported, **–relax** is accepted, but has no effect.

–rpath *directory*

Add a directory to the runtime library search path. This is used when linking an ELF executable with shared objects. All **–rpath** arguments are concatenated and passed to the runtime linker, which uses them to locate shared objects at runtime. The **–rpath** option is also used when locating shared objects which are needed by shared objects explicitly included in the link; see the description of the **–rpath–link** option. If **–rpath** is not used when linking an ELF executable, the contents of the environment variable **LD_RUN_PATH** will be used if it is defined.

The **–rpath** option may also be used on SunOS. By default, on SunOS, the linker will form a runtime search patch out of all the **–L** options it is given. If a **–rpath** option is used, the runtime search path will be formed exclusively using the **–rpath** options, ignoring the **–L** options. This can be useful when using gcc, which adds many **–L** options which may be on NFS mounted filesystems.

–rpath–link *directory*

When using ELF or SunOS, one shared library may require another. This happens when an *ld –shared* link includes a shared library as one of the input files.

When the linker encounters such a dependency when doing a non-shared, non-relocateable link, it will automatically try to locate the required shared library and include it in the link, if it is not included explicitly. In such a case, the **–rpath–link** option specifies the first set of directories to search. The **–rpath–link** option may specify a sequence of directory names either by specifying a list of names separated by colons, or by appearing multiple times.

If the required shared library is not found, the linker will issue a warning and continue with the link.

–S Omits debugger symbol information (but not all symbols) from the output file.

–s Omits all symbol information from the output file.

–shared Create a shared library. This is currently only supported on ELF and SunOS platforms (on SunOS it is not required, as the linker will automatically create a shared library when there are undefined symbols and the **–e** option is not used).

–sort–common

Normally, when *ld* places the global common symbols in the appropriate output sections, it sorts them by size. First come all the one byte symbols, then all the two bytes, then all the four bytes, and then everything else. This is to prevent gaps between symbols due to alignment constraints. This option disables that sorting.

–split–by–reloc *count*

Trys to creates extra sections in the output file so that no single output section in the file contains more than *count* relocations. This is useful when generating huge relocatable for downloading into certain real time kernels with the COFF object file format; since COFF cannot represent more than 65535 relocations in a single section. Note that this will fail to work with object file formats which do not support arbitrary sections. The linker will not split up individual input sections for redistribution, so if a single input section contains more than *count* relocations one output section will contain that many relocations.

–split–by–file

Similar to **–split–by–reloc** but creates a new output section for each input file.

–Tbss *org*
–Tdata *org*
–Ttext *org*

Use *org* as the starting address for—respectively—the **bss**, **data**, or the **text** segment of the output file. *textorg* must be a hexadecimal integer.

–T *commandfile*
Equivalent to **–c** *commandfile*; supported for compatibility with other tools.

–t Prints names of input files as *ld* processes them.

–u *sym* Forces *sym* to be entered in the output file as an undefined symbol. This may, for example, trigger linking of additional modules from standard libraries. **–u** may be repeated with different option arguments to enter additional undefined symbols.

–Ur For anything other than C++ programs, this option is equivalent to **–r**: it generates relocatable output—i.e., an output file that can in turn serve as input to *ld*. When linking C++ programs, **–Ur** *will* resolve references to constructors, unlike **–r**.

––verbose
Display the version number for *ld* and list the supported emulations. Display which input files can and can not be opened.

–v Display the version number for *ld*.

–V Display the version number and the supported emulations for *ld*.

––version
Display the version number for *ld* and exit.

–warn–common
Warn when a common symbol is combined with another common symbol or with a symbol definition. Traditional linkers allow this somewhat sloppy practice, but other linkers may not not. This option allows you to find potential problems from combining global symbols.

–warn–constructors
Warn if any global constructors are used. This is only useful for a few object file formats. For formats like COFF or ELF, the linker can not detect the use of global constructors.

–warn–multiple–gp Warn if the output file requires multiple global-pointer values. This option is only meaningful for certain processors, such as the Alpha.

–warn–once
Only warn once for each undefined symbol, rather than once per module which refers to it.

–warn–section–align
Warn if the address of an output section is changed because of alignment. Typically, the alignment will be set by an input section. The address will only be changed if it not explicitly specified; that is, if the SECTIONS command does not specify a start address for the section.

––whole–archive
For each archive mentioned on the command line after the **—whole–archive** option, include every object file in the archive in the link, rather than searching the archive for the required object files. This is normally used to turn an archive file into a shared library, forcing every object to be included in the resulting shared library.

––no–whole–archive
Turn off the effect of the **—whole–archive** option for archives which appear later on the command line.

––wrap *symbol*
Use a wrapper function for *symbol*. Any undefined reference to *symbol* will be resolved to **__wrap_***symbol***.** Any undefined reference to **__real_***symbol* will be resolved to *symbol*.

–X Delete all temporary local symbols. For most targets, this is all local symbols whose names begin with **L**.

–x Delete all local symbols.

ENVIRONMENT

You can change the behavior of *ld* with the environment variable *GNUTARGET*.

GNUTARGET determines the input-file object format if you don't use **–b** (or its synonym **–format**). Its value should be one of the BFD names for an input format. If there is no *GNUTARGET* in the environment, *ld* uses the natural format of the host. If *GNUTARGET* is set to "default" then BFD attempts to discover the input format by examining binary input files; this method often succeeds, but there are potential ambiguities, since there is no method of ensuring that the magic number used to flag object-file formats is unique. However, the configuration procedure for BFD on each system places the conventional format for that system first in the search-list, so ambiguities are resolved in favor of convention.

COPYING

Copyright (c) 1991, 1992 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

SEE ALSO

objdump

ld and *binutils* entries in **info**

ld: the GNU linker, Steve Chamberlain and Roland Pesch; *The GNU Binary Utilities*, Roland H. Pesch.

NAME

lex – fast lexical analyzer generator

SYNOPSIS

```
lex [[-78BbcdFfhIiLlnpsTtVvw] [-C[aefFmr]] [-Pprefix]
    [-Sskeleton] [filename ...]
```

DESCRIPTION

lex is a tool for generating *scanners*: programs which recognized lexical patterns in text. *lex* reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called *rules*. *Lex* generates as output a C source file, **lex.yy.c,** which defines a routine *yylex*( ). This file is compiled and linked with the **–ll** library to produce an executable. When the executable is run, it analyzes its input for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

For full documentation, see *lexdoc*. This manual entry is intended for use as a quick reference.

lex has the following options:

–7 Generate a 7-bit scanner, which can save considerable table space, especially when using **–Cf** or **–CF** (and, at most sites, **–7** is on by default for these options. To see if this is the case, use the **–v** verbose flag and check the flag summary it reports).

–8 Generate an 8-bit scanner. This is the default except for the **–Cf** and **–CF** compression options, for which the default is site-dependent, and can be checked by inspecting the flag summary generated by the **–v** option.

–B Generate a *batch* scanner instead of an *interactive* scanner (see **–I** below). See *lexdoc* for details. Scanners using **–Cf** or **–CF** compression options automatically specify this option, too.

–b Generate backing-up information to *lex.backup*. This is a list of scanner states which require backing up and the input characters on which they do so. By adding rules one can remove backing-up states. If all backing-up states are eliminated and **–Cf** or **–CF** is used, the generated scanner will run faster.

–C[**aefFmr**] Specify degree of table compression and scanner optimization.

–Ca Trade off larger tables in the generated scanner for faster performance because the elements of the tables are better aligned for memory access and computation. This option can double the size of the tables used by your scanner.

–Ce Construct *equivalence* classes, i.e., sets of characters which have identical lexical properties. Equivalence classes usually give dramatic reductions in the final table / object file sizes (typically a factor of 2-5) and are pretty cheap performance-wise (one array look-up per character scanned).

–Cf Generate *full* scanner tables — *lex* should not compress the tables by taking advantages of similar transition functions for different states.

–CF Use the alternate fast scanner representation (described in *lexdoc*).

–Cm Construct *meta-equivalence classes*, which are sets of equivalence classes (or characters, if equivalence classes are not being used) that are commonly used together. Meta-equivalence classes are often a big win when using compressed tables, but they have a moderate performance impact (one or two "if" tests and one array look-up per character scanned).

–Cr *Bypass* using stdio for input in generated scanner. In general this option results in a minor performance gain only worthwhile if used in conjunction with **–Cf** or **–CF**. It can cause surprising behavior if you use stdio yourself to read from **yyin** prior to calling the scanner.

–C Alone, compress scanner tables but use neither equivalence classes nor meta-equivalence classes.

The options **–Cf** or **–CF** and **–Cm** do not make sense together — there is no opportunity for meta-equivalence classes if the table is not being compressed. Otherwise the options may be freely mixed.

The default setting is **–Cem**, which specifies that *lex* should generate equivalence classes and meta-equivalence classes. This setting provides the highest degree of table compression. You can trade off faster-executing scanners at the cost of larger tables with the following generally being true:

slowest & smallest	**–Cem**
	–Cm
	–Ce
	–C
	–C{f,F}e
	–C{f,F}
fastest & largest	**–C{f,F}a**

–C options are cumulative.

–c Does nothing; a deprecated option included for POSIX compliance.

NOTE: in previous releases of *lex* **–c** specified table-compression options. This functionality is now given by the **–C** flag. To ease the the impact of this change, when *lex* encounters **–c**, it currently issues a warning message and assumes that **–C** was desired instead. In the future this "promotion" of **–c** to **–C** will go away in the name of full POSIX compliance (unless the POSIX meaning is removed first).

–d Run the generated scanner in *debug* mode. Whenever a pattern is recognized and the global *yy_flex_debug* is non-zero (which is the default), the scanner will write to stderr a line of the form:

```
--accepting rule at line 53 ("the matched text")
```

The line number refers to the location of the rule in the file defining the scanner (i.e., the file that was fed to lex). Messages are also generated when the scanner backs up, accepts the default rule, reaches the end of its input buffer (or encounters a NUL; the two look the same as far as the scanner's concerned), or reaches an end-of-file.

–F Use *fast* scanner table representation (and bypass stdio). This representation is about as fast as the full table representation (**–f)**, and for some sets of patterns will be considerably smaller (and for others, larger). See *lexdoc* for more details.

This option is equivalent to **–CFr** (see below).

–f Use *fast scanner*. No table compression is done and stdio is bypassed. The result is large but fast. This option is equivalent to **–Cfr** (see below).

–h Generate a "help" summary of *lex's* options to stderr and then exits.

–I Generate an *interactive* scanner, that is, a scanner which stops immediately rather than looking ahead if it knows that the currently scanned text cannot be part of a longer rule's match. This is the opposite of *batch* scanners (see **–B** above). See *lexdoc* for details.

Note, **–I** cannot be used in conjunction with *full* or *fast tables*, i.e., the **–f**, **–F**, **–Cf**, or **–CF** flags. For other table compression options, **–I** is the default.

–i Generate a *case-insensitive* scanner. The case of letters given in the *lex* input patterns will be ignored, and tokens in the input will be matched regardless of case. The matched text given in *yytext* will have the preserved case (i.e., it will not be folded).

–L Do *note* generate **#line** directives in **lex.yy.c**. The default is to generate such directives so error messages in the actions will be correctly located with respect to the original *lex* input file, and not to the fairly meaningless line numbers of **lex.yy.c**.

–l Turn on maximum compatibility with the original AT&T lex implementation, at a considerable performance cost. This option is incompatible with **–f**, **–F**, **–Cf**, or **–CF.** See *lexdoc* for details.

–n Does nothing. Another deprecated option included only for POSIX compliance.

–P*prefix*

Change the default *yy* prefix used by *lex* to be *prefix* instead. See *lexdoc* for a description of all the global variables and file names that this affects.

–p Generate a performance report to stderr. The report consists of comments regarding features of the *lex* input file which will cause a loss of performance in the resulting scanner. If you give the flag twice, you will also get comments regarding features that lead to minor performance losses.

–S*skeleton_file*
Use *skeleton_file* to construct the scanner instead of the default file. You'll never need this option unless you are doing *lex* maintenance or development.

–s Suppress the *default rule* (that unmatched scanner input is echoed to stdout). If the scanner encounters input that does not match any of its rules, it aborts with an error.

–T Run in *trace* mode. It will generate a lot of messages to stderr concerning the form of the input and the resultant non-deterministic and deterministic finite automata. This option is mostly for use in maintaining *lex*.

–t Write the scanner it generates to standard output instead of **lex.yy.c**.

–V Print the version number to stderr and exits.

–v Write to stderr a summary of statistics regarding the scanner it generates.

–w Suppress warning messages.

SUMMARY OF LEX REGULAR EXPRESSIONS

The patterns in the input are written using an extended set of regular expressions. These are:

Pattern	**Matches**
x	match the character 'x'
.	any character except newline
[xyz]	a "character class"; in this case, the pattern matches either an 'x', a 'y', or a 'z'
[abj-oZ]	a "character class" with a range in it; matches an 'a', a 'b', any letter from 'j' through 'o', or a 'Z'
[^A-Z]	a "negated character class", i.e., any character but those in the class. In this case, any character *except* an uppercase letter.
[^A-Z \n]	any character *except* an uppercase letter or a newline
*r**	zero or more *r*'s, where *r* is any regular expression
r+	one or more *r*'s
r?	zero or one *r*'s (that is, "an optional *r*")
r{2,5}	anywhere from two to five *r*'s
r{2,}	two or more *r*'s
r{4}	exactly 4 *r*'s
{*name*}	the expansion of the *name* definition (see above)

Pattern	Matches
[xyz]"foo"	literal string: [xyz]"foo
\X	if X is an 'a', 'b', 'f', 'n', 'r', 't', or 'v', then the ANSI-C interpretation of \x. Otherwise, a literal 'X' (used to escape operators such as '*')
\123	the character with octal value 123
\x2a	the character with hexadecimal value 2a
(r)	match an *r*; parentheses are used to override precedence (see below)
rs	"concatenation:" the regular expression *r* followed by the regular expression *s*
r|s	either an *r* or an *s*
r/s	an *r* but only if it is followed by an *s*. The *s* is not part of the matched text. This type of pattern is called as "trailing context".
^r	an *r*, but only at the beginning of a line
r$	an *r*, but only at the end of a line. Equivalent to "r/\n".
<s>r	an *r*, but only in start condition *s* (see below for discussion of start conditions)
<s1,s2,s3>r	same, but in any of start conditions *s1*, *s2*, or *s3*
<*>r	an *r* in any start condition, even an exclusive one.
<<EOF>>	an end-of-file
<s1,s2><<EOF>>	an end-of-file when in start condition *s1* or *s2*

The regular expressions listed above are grouped according to precedence, from highest precedence at the top to lowest at the bottom. Those grouped together have equal precedence.

Some notes on patterns:

- Negated character classes *match newlines* unless "\n" (or an equivalent escape sequence) is one of the characters explicitly present in the negated character class (e.g., "[^A-Z\n]").
- A rule can have at most one instance of trailing context (the '/' operator or the '$' operator). The start condition, '^', and "<<EOF>>" patterns can only occur at the beginning of a pattern, and, as well as with '/' and '$', cannot be grouped inside parentheses. The following are all illegal:

```
foo/bar$
foo|(bar$)
foo|^bar
<sc1>foo<sc2>bar
```

SUMMARY OF SPECIAL ACTIONS

In addition to arbitrary C code, the following can appear in actions:

ECHO copies yytext to the scanner's output.
BEGIN followed by the name of a start condition places the scanner in the

corresponding start condition.

REJECT
directs the scanner to proceed on to the "second best" rule which matched the input (or a prefix of the input). *yytext* and *yyleng* are set up appropriately. Note that **REJECT** is a particularly expensive feature in terms scanner performance; if it is used in *any* of the scanner's actions it will slow down *all* of the scanner's matching. Furthermore, **REJECT** cannot be used with the **–f** or **–F** options. Note also that unlike the other special actions, **REJECT** is a *branch;* code immediately following it in the action will *not* be executed.

yymore()
tells the scanner that the next time it matches a rule, the corresponding token should be *appended* onto the current value of *yytext* rather than replacing it.

yyless(*n*)
returns all but the first *n* characters of the current token back to the input stream, where they will be rescanned when the scanner looks for the next match. *yytext* and *yyleng* are adjusted appropriately (e.g., *yyleng* will now be equal to *n*).

unput(*c*)
puts the character *c* back onto the input stream. It will be the next character scanned.

input() reads the next character from the input stream (this routine is called *yyinput()* if the scanner is compiled using **C++**).

yyterminate()
can be used in lieu of a return statement in an action. It terminates the scanner and returns a 0 to the scanner's caller, indicating "all done". By default, *yyterminate*( ) is also called when an end-of-file is encountered. It is a macro and may be redefined.

YY_NEW_FILE
is an action available only in <<EOF>> rules. It means "Okay, I've set up a new input file, continue scanning". It is no longer required; you can just assign *yyin* to point to a new file in the <<EOF>> action.

yy_create_buffer(*file*, *size*)
takes a **FILE** pointer and an integer *size*. It returns a YY_BUFFER_STATE handle to a new input buffer large enough to accommodate *size* characters and associated with the given file. When in doubt, use YY_BUF_SIZE for the size.

yy_switch_to_buffer new_buffer
switches the scanner's processing to scan for tokens from the given buffer, which must be a YY_BUFFER_STATE.

yy_delete_buffer(*buffer*)
deletes the given buffer.

VALUES AVAILABLE TO THE USER

The following values are available to the user:

**char *****yytext*
holds the text of the current token. It may be modified but not lengthened (you cannot append characters to the end). Modifying the last character may affect the activity of rules anchored using '^' during the next scan; see *lexdoc* for details. If the special directive **%array**

appears in the first section of the scanner description, then *yytext* is instead declared

```
char yytext[YYLMAX]
```

where YYLMAX is a macro definition that you can redefine in the first section if you don't like the default value (generally 8K bytes). Using **%array** results in somewhat slower scanners, but the value of *yytext* becomes immune to calls to *input*( ) and *unput*( ), which potentially destroy its value when *yytext* is a character pointer. The opposite of **%array** is **%pointer,** which is the default.

int *yyleng*
holds the length of the current token.

**FILE ***_yyin_
is the file which by default *lex* reads from. It may be redefined but doing so only makes sense before scanning begins or after an EOF has been encountered. Changing it in the midst of scanning will have unexpected results since *lex* buffers its input; use *yyrestart*( ) instead. Once scanning terminates because an end-of-file has been seen, you can assign *yyin* at the new input file and then call the scanner again to continue scanning.

void *yyrestart*(**FILE ***_new_file_)
may be called to point *yyin* at the new input file. The switch-over to the new file is immediate (any previously buffered-up input is lost). Note that calling *yyrestart*( ) with *yyin* as an argument thus throws away the current input buffer and continues scanning the same input file.

**FILE ***_yyout_
is the file to which **ECHO** actions are done. It can be reassigned by the user.

YY_CURRENT_BUFFER
returns a **YY_BUFFER_STATE** handle to the current buffer.

YY_START
returns an integer value corresponding to the current start condition. You can subsequently use this value with **BEGIN** to return to that start condition.

MACROS AND FUNCTIONS YOU CAN REDEFINE

Lex allows you to redefine the following macros and functions:

YY_DECL
controls how the scanning routine is declared. By default, it is "int yylex()", or, if prototypes are being used, "int yylex(void)". This definition may be changed by redefining the "YY_DECL" macro. Note that if you give arguments to the scanning routine using a K&R-style / non-prototyped function declaration, you must terminate the definition with a semi-colon (;).

YY_INPUT
The nature of how the scanner gets its input can be controlled by redefining the *YY_INPUT* macro. YY_INPUT's calling sequence is

```
YY_INPUT(buf,result,max_size)
```

Its action is to place up to *max_size* characters in the character array

buf and return in the integer variable *result* either the number of characters read or the constant YY_NULL (traditionally 0) to indicate EOF. The default YY_INPUT reads from the global file-pointer *yyin*. A sample redefinition of YY_INPUT (in the definitions section of the input file):

```
%{
#undef YY_INPUT
#define YY_INPUT(buf,result,max_size) \
    { \
    int c = getchar(); \
    result = (c == EOF) ? YY_NULL : (buf[0] = c, 1); \
    }
%}
```

yywrap()
: When the scanner receives an end-of-file indication from YY_INPUT, it then checks the function *yywrap*( ) function. If *yywrap*( ) returns false (zero), then it is assumed that the function has gone ahead and set up *yyin* to point to another input file, and scanning continues. If it returns true (non-zero), then the scanner terminates, returning 0 to its caller. The default *yywrap*( ) always returns 1.

YY_USER_ACTION
: can be redefined to provide an action which is always executed prior to the matched rule's action.

YY_USER_INIT
: This macro may be redefined to provide an action which is always executed before the first scan.

YY_BREAK
: In the generated scanner, the actions are all gathered in one large switch statement and separated using YY_BREAK, which may be redefined. By default, it is simply a "break", to separate each rule's action from the following rule's.

FILES

–ll library with which scanners may be linked to obtain default versions of *yywrap*( ) and *main*( ).

lex.yy.c generated scanner (called **lexyy.c** on some systems).

lex.backup
: backing-up information for **–b** flag (called **lex.bck** on some systems).

SEE ALSO

awk, *lexdoc*, *sed*, *yacc*

M. E. Lesk and E. Schmidt, *LEX – Lexical Analyzer Generator*

DIAGNOSTICS

Lex can generate the following diagnostic messages

reject_used_but_not_detected undefined
: The scanner uses **REJECT** but that *lex* failed to find it in the first two sections. This may happen if you use a **#include** file to insert it. Make an explicit reference to the action in your *lex* input file. (Note that previously *lex* supported a **%used**/**%unused** mechanism for dealing

with this problem; this feature is still supported but now deprecated, and will go away soon unless the author hears from people who can argue compellingly that they need it.)

yymore_used_but_not_detected
: The scanner uses *yymore*( ) but *lex* failed to find it in the first two sections. This may happen if you use a **#include** file to insert it. Make an explicit reference to the action in your *lex* input file. (Note that previously *lex* supported a **%used**/**%unused** mechanism for dealing with this problem; this feature is still supported but now deprecated, and will go away soon unless the author hears from people who can argue compellingly that they need it.)

lex scanner jammed
: A scanner compiled with **–s** has encountered an input string which wasn't matched by any of its rules.

warning, rule cannot be matched
: The given rule cannot be matched because it follows other rules that will always match the same text as it. See *lexdoc* for an example.

warning, –s option given but default rule can be matched
: It is possible (perhaps only in a particular start condition) that the default rule (match any single character) is the only one that will match a particular input. Since **–s** was given, presumably this is not intended.

scanner input buffer overflowed
: A scanner rule matched more text than the available dynamic memory.

token too large, exceeds YYLMAX
: Your scanner uses **%array** and one of its rules matched a string longer than the YYLMAX constant (8K bytes by default). You can increase the value by **#define**'ing YYLMAX in the definitions section of your *lex* input.

scanner requires –8 flag to use the character 'x'
: Your scanner specification includes recognizing the 8-bit character *'x'* and you did not specify the **–8** flag, and your scanner defaulted to 7-bit because you used the **–Cf** or **–CF** table compression options.

lex scanner push-back overflow
: you used *unput*( ) to push back so much text that the scanner's buffer could not hold both the pushed-back text and the current token in `yytext`. Ideally the scanner should dynamically resize the buffer in this case, but at present it does not.

input buffer overflow, can't enlarge buffer because scanner uses REJECT
: the scanner was working on matching an extremely large token and needed to expand the input buffer. This doesn't work with scanners that use **REJECT**.

fatal lex scanner internal error--end of buffer missed
: This can occur in an scanner which is reentered after a long-jump has jumped out (or over) the scanner's activation frame. Before reentering the scanner, use:

```
yyrestart( yyin );
```

AUTHOR

Vern Paxson, with the help of many ideas and much inspiration from Van

Jacobson. Original version by Jef Poskanzer.

See *lexdoc* for additional credits and the address to send comments to.

DEFICIENCIES / BUGS

Some trailing context patterns cannot be properly matched and generate warning messages ("dangerous trailing context"). These are patterns where the ending of the first part of the rule matches the beginning of the second part, such as "zx*/xy*", where the 'x*' matches the 'x' at the beginning of the trailing context. (Note that the POSIX draft states that the text matched by such patterns is undefined.)

For some trailing context rules, parts which are actually fixed-length are not recognized as such, leading to the performance loss already mentioned. In particular, parts using '|' or {n} (such as "foo{3}") are always considered variable-length.

Combining trailing context with the special '|' action can result in *fixed* trailing context being turned into the more expensive *variable* trailing context. For example, in the following:

```
%%
abc      |
xyz/def
```

Use of *unput*( ) or *input*( ) invalidates yytext and yyleng, unless the **%array** directive or the **–l** option has been used.

Use of *unput*( ) to push back more text than was matched can result in the pushed-back text matching a beginning-of-line ('^') rule even though it didn't come at the beginning of the line (though this is rare!).

Pattern-matching of NUL's is substantially slower than matching other characters.

Dynamic resizing of the input buffer is slow, as it entails rescanning all the text matched so far by the current (generally huge) token.

lex does not generate correct #line directives for code internal to the scanner; thus, bugs in its skeleton file yield bogus line numbers.

Due to both buffering of input and read-ahead, you cannot intermix calls to **<stdio.h>** routines, such as, for example, *getchar()*( ), with *lex* rules and expect it to work. Call *input()*( ) instead.

The total table entries listed by the **–v** flag excludes the number of table entries needed to determine what rule has been matched. The number of entries is equal to the number of DFA states if the scanner does not use **REJECT**, and somewhat greater than the number of states if it does.

REJECT cannot be used with the **–f** or **–F** options.

The *lex* internal algorithms need documentation.

NAME

lexdoc – documentation for lex, fast lexical analyzer generator

SYNOPSIS

`lex [-78BbcdFfhIiLlnpsTtVvw] [-C[aefFmr]] [-P`*`prefix`*`] [-S`*`skeleton`*`] [`*`filename ...`*`]`

DESCRIPTION

lex is a tool for generating *scanners*: programs which recognized lexical patterns in text. *lex* reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called *rules*. *Lex* generates as output a C source file, **lex.yy.c**, which defines a routine *yylex*( ). This file is compiled and linked with the **–ll** library to produce an executable. When the executable is run, it analyzes its input for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

SOME SIMPLE EXAMPLES

First some simple examples to get the flavor of how one uses *lex*. The following *lex* input specifies a scanner which whenever it encounters the string "username" will replace it with the user's login name:

```
%%
username    printf( "%s", getlogin() );
```

By default, any text not matched by a *lex* scanner is copied to the output, so the net effect of this scanner is to copy its input file to its output with each occurrence of "username" expanded. In this input, there is just one rule. "username" is the *pattern* and the "printf" is the *action*. The "%%" marks the beginning of the rules.

Here's another simple example:

```
        int num_lines = 0, num_chars = 0;

%%
\n      ++num_lines; ++num_chars;

%%
main()
        {
        yylex();
        printf( "# of lines = %d, # of chars = %d\n",
                num_lines, num_chars );
        }
```

This scanner counts the number of characters and the number of lines in its input (it produces no output other than the final report on the counts). The first line declares two globals, "num_lines" and "num_chars", which are accessible both inside *yylex*( ) and in the *main*( ) routine declared after the second "%%". There are two rules, one which matches a newline ("\n") and increments both the line count and the character count, and one which matches any character other than a newline (indicated by the "." regular expression).

A somewhat more complicated example:

```
/* scanner for a toy Pascal-like language */

%{
/* need this for the call to atof() below */
#include <math.h>
%}

DIGIT    [0-9]
ID       [a-z][a-z0-9]*

%%

{DIGIT}+    {
            printf( "An integer: %s (%d)\n", yytext,
                    atoi( yytext ) );
            }

{DIGIT}+"."{DIGIT}*        {
            printf( "A float: %s (%g)\n", yytext,
                    atof( yytext ) );
            }

if|then|begin|end|procedure|function        {
            printf( "A keyword: %s\n", yytext );
            }

{ID}        printf( "An identifier: %s\n", yytext );

"+"|"-"|"*"|"/"   printf( "An operator: %s\n", yytext );

"{"[^}\n]*"}"     /* eat up one-line comments */

[ \t\n]+          /* eat up whitespace */


%%

main( argc, argv )
int argc;
char **argv;
    {
    ++argv, --argc;  /* skip over program name */
    if ( argc > 0 )
            yyin = fopen( argv[0], "r" );
    else
            yyin = stdin;

    yylex();
    }
```

This is the beginnings of a simple scanner for a language like Pascal. It identifies different types of *tokens* and reports on what it has seen.

The details of this example will be explained in the following sections.

FORMAT OF THE INPUT FILE

The *lex* input file consists of three sections, separated by a line with just **%%** in it:

definitions
%%
rules
%%
user code

The *definitions* section contains declarations of simple *name* definitions to simplify the scanner specification, and declarations of *start conditions*, which are explained in a later section.

Name definitions have the form:

name definition

The *name* is a word beginning with a letter or an underscore ('_') followed by zero or more letters, digits, '_', or '-' (dash). The definition is taken to begin at the first non-white-space character following the name and continuing to the end of the line. The definition can subsequently be referred to using "{name}", which will expand to "(definition)". For example,

```
DIGIT    [0-9]
ID       [a-z][a-z0-9]*
```

defines "DIGIT" to be a regular expression which matches a single digit, and "ID" to be a regular expression which matches a letter followed by zero-or-more letters-or-digits. A subsequent reference to

```
{DIGIT}+"."{DIGIT}*
```

is identical to

```
([0-9])+"."([0-9])*
```

and matches one-or-more digits followed by a '.' followed by zero-or-more digits.

The *rules* section of the *lex* input contains a series of rules of the form:

pattern action

where the pattern must be unindented and the action must begin on the same line.

See below for a further description of patterns and actions.

Finally, the user code section is simply copied to **lex.yy.c** verbatim. It is used for companion routines which call or are called by the scanner. The presence of this section is optional; if it is missing, the second **%%** in the input file may be skipped, too.

In the definitions and rules sections, any *indented* text or text enclosed in **%{** and **%}** is copied verbatim to the output (with the **%{}**'s removed). The **%{}**'s must

appear unindented on lines by themselves.

In the rules section, any indented or **%{}** text appearing before the first rule may be used to declare variables which are local to the scanning routine and (after the declarations) code which is to be executed whenever the scanning routine is entered. Other indented or %{} text in the rule section is still copied to the output, but its meaning is not well-defined and it may well cause compile-time errors (this feature is present for POSIX compliance; see below for other such features).

In the definitions section (but not in the rules section), an unindented comment (i.e., a line beginning with "/*") is also copied verbatim to the output up to the next "*/".

PATTERNS

The patterns in the input are written using an extended set of regular expressions. These are:

Pattern	**Matches**
x	match the character 'x'
.	any character except newline
[xyz]	a "character class"; in this case, the pattern matches either an 'x', a 'y', or a 'z'
[abj-oZ]	a "character class" with a range in it; matches an 'a', a 'b', any letter from 'j' through 'o', or a 'Z'
[^A-Z]	a "negated character class", i.e., any character but those in the class. In this case, any character *except* an uppercase letter.
[^A-Z\n]	any character *except* an uppercase letter or a newline
*r**	zero or more *r*'s, where *r* is any regular expression
r+	one or more *r*'s
r?	zero or one *r*'s (that is, an optional *r*)
r{2,5}	anywhere from two to five *r*'s
r{2,}	two or more *r*'s
r{4}	exactly 4 *r*'s
{*name*}	the expansion of the *name* definition (see above)
"[xyz]\"foo"	the literal string: [xyz]"foo
X	If *X* is an 'a', 'b', 'f', 'n', 'r', 't', or 'v', then the ANSI C interpretation of \x. Otherwise, a literal 'X' (used to escape operators such as '*') \123the character with octal value 123 \x2athe character with hexadecimal value 2a (*r*)match an *r*; parentheses are used to override precedence (see below)
*rs*T{	"concatenation:" the regular expression *r*

Pattern	Matches
	followed by the regular expression *s*
r|s	either an *r* or an *s*
r/s	an *r* but only if it is followed by an *s*. The *s* is not part of the matched text. This type of pattern is called as "trailing context".
^r	an *r*, but only at the beginning of a line
r$	an *r*, but only at the end of a line. Equivalent to "r/\n".
<s>r	an *r*, but only in start condition *s* (see below for discussion of start conditions)
<s1,s2,s3>r	same, but in any of start conditions *s1*, *s2*, or *s3*
<*>*r*	an *r* in any start condition, even an exclusive one.
<<EOF>>	an end-of-file
<s1,s2><<EOF>>	an end-of-file when in start condition *s1* or *s2*

Note that inside of a character class, all regular expression operators lose their special meaning except escape ('\') and the character class operators, '-', ']', and, at the beginning of the class, '^'.

The regular expressions listed above are grouped according to precedence, from highest precedence at the top to lowest at the bottom. Those grouped together have equal precedence. For example,

```
foo|bar*
```

is the same as

```
(foo)|(ba(r*))
```

since the '*' operator has higher precedence than concatenation, and concatenation higher than alternation ('|'). This pattern therefore matches *either* the string "foo" *or* the string "ba" followed by zero-or-more r's. To match "foo" or zero-or-more "bar"'s, use:

```
foo|(bar)*
```

and to match zero-or-more "foo"'s-or-"bar"'s:

```
(foo|bar)*
```

Some notes on patterns:

- A negated character class such as the example "[^A-Z]" above *will match a newline* unless "\n" (or an equivalent escape sequence) is one of the characters explicitly present in the negated character class (e.g., "[^A-Z\n]"). This is unlike how many other regular expression tools treat negated character classes, but unfortunately the inconsistency is historically entrenched. Matching newlines means that a pattern like [^"]* can match the entire input unless there's another quote in the input.

- A rule can have at most one instance of trailing context (the '/' operator or the '$' operator). The start condition, '^', and "<<EOF>>" patterns can only occur at the beginning of a pattern, and, as well as with '/' and '$', cannot be grouped inside parentheses. A '^' which does not occur at the beginning of a rule or a '$' which does not occur at the end of a rule loses its special properties and is treated as a normal character.
- The following are illegal:

```
foo/bar$
<sc1>foo<sc2>bar
```

  Note that the first of these, can be written "foo/bar\n".

- The following will result in '$' or '^' being treated as a normal character:

```
foo|(bar$)
foo|^bar
```

  If what's wanted is a "foo" or a bar-followed-by-a-newline, the following could be used (the special '|' action is explained below):

```
foo      |
bar$     /* action goes here */
```

  A similar trick will work for matching a foo or a bar-at-the-beginning-of-a-line.

HOW THE INPUT IS MATCHED

When the generated scanner is run, it analyzes its input looking for strings which match any of its patterns. If it finds more than one match, it takes the one matching the most text (for trailing context rules, this includes the length of the trailing part, even though it will then be returned to the input). If it finds two or more matches of the same length, the rule listed first in the *lex* input file is chosen.

Once the match is determined, the text corresponding to the match (called the I token) is made available in the global character pointer *yytext*, and its length in the global integer *yyleng*. The *action* corresponding to the matched pattern is then executed (a more detailed description of actions follows), and then the remaining input is scanned for another match.

If no match is found, then the *default rule* is executed: the next character in the input is considered matched and copied to the standard output. Thus, the simplest legal *lex* input is:

```
%%
```

which generates a scanner that simply copies its input (one character at a time) to its output.

Note that *yytext* can be defined in two different ways: either as a character *pointer* or as a character *array*. You can control which definition *lex* uses by including one of the special directives **%pointer** or **%array** in the first (definitions) section of your lex input. The default is **%pointer**, unless you use the **–l** AT&T lex compatibility option, in which case *yytext* will be an array. The advantage of using **%pointer** is substantially faster scanning and no buffer overflow when matching very large tokens (unless you run out of dynamic

memory). The disadvantage is that you are restricted in how your actions can modify *yytext* (see the next section), and calls to the *input*( ) and *unput*( ) functions destroy the present contents of *yytext*, which can be a considerable porting headache when moving between different *lex* versions.

The advantage of **%array** is that you can then modify *yytext* to your heart's content, and calls to *input*( ) and *unput*( ) do not destroy *yytext* (see below). Furthermore, existing AT&T *lex* programs sometimes access *yytext* externally using declarations of the form:

```
extern char yytext[];
```

This definition is erroneous when used with **%pointer**, but correct for **%array**.

%array defines *yytext* to be an array of YYLMAX characters, which defaults to a fairly large value. You can change the size by simply **#define'ing** YYLMAX to a different value in the first section of your *lex* input. As mentioned above, with **%pointer** *yytext* grows dynamically to accommodate large tokens. While this means your **%pointer** scanner can accommodate very large tokens (such as matching entire blocks of comments), bear in mind that each time the scanner must resize *yytext* it also must rescan the entire token from the beginning, so matching such tokens can prove slow. *yytext* presently does *not* dynamically grow if a call to *unput*( ) results in too much text being pushed back; instead, a run-time error results.

ACTIONS

Each pattern in a rule has a corresponding action, which can be any arbitrary C statement. The pattern ends at the first non-escaped whitespace character; the remainder of the line is its action. If the action is empty, then when the pattern is matched the input token is simply discarded. For example, here is the specification for a program which deletes all occurrences of "zap me" from its input:

```
%%
"zap me"
```

(It will copy all other characters in the input to the output since they will be matched by the default rule.)

Here is a program which compresses multiple blanks and tabs down to a single blank, and throws away whitespace found at the end of a line:

```
%%
[ \t]+        putchar( ' ' );
[ \t]+$       /* ignore this token */
```

If the action contains a '{', then the action spans till the balancing '}' is found, and the action may cross multiple lines. *lex* knows about C strings and comments and won't be fooled by braces found within them, but also allows actions to begin with **%{** and will consider the action to be all the text up to the next **%}** (regardless of ordinary braces inside the action).

An action consisting solely of a vertical bar ('|') means "same as the action for the next rule." See below for an illustration.

Actions can include arbitrary C code, including **return** statements to return a value to whatever routine called *yylex*( ). Each time *yylex*( ) is called it

continues processing tokens from where it last left off until it either reaches the end of the file or executes a return.

Actions are free to modify *yytext* except for lengthening it (adding characters to its end--these will overwrite later characters in the input stream). Modifying the final character of *yytext* may alter whether when scanning resumes rules anchored with '^' are active. Specifically, changing the final character of *yytext* to a newline will activate such rules on the next scan, and changing it to anything else will deactivate the rules. Users should not rely on this behavior being present in future releases. Finally, note that none of this paragraph applies when using **%array** (see above).

Actions are free to modify *yyleng* except they should not do so if the action also includes use of *yymore*( ) (see below).

There are a number of special directives which can be included within an action:

ECHO copies *yytext* to the scanner's output.

BEGIN followed by the name of a start condition places the scanner in the corresponding start condition (see below).

REJECT

directs the scanner to proceed on to the "second best" rule which matched the input (or a prefix of the input). The rule is chosen as described above in "How the Input is Matched", and *yytext* and *yyleng* set up appropriately. It may either be one which matched as much text as the originally chosen rule but came later in the *lex* input file, or one which matched less text. For example, the following will both count the words in the input and call the routine *special*( ) whenever "frob" is seen:

```
        int word_count = 0;
%%

frob        special(); REJECT;
[^ \t\n]+   ++word_count;
```

Without the **REJECT**, any "frob"'s in the input would not be counted as words, since the scanner normally executes only one action per token. Multiple **REJECT**'s are allowed, each one finding the next best choice to the currently active rule. For example, when the following scanner scans the token "abcd", it will write "abcdabcaba" to the output:

```
%%
a        |
ab       |
abc      |
abcd     ECHO; REJECT;
.|\n     /* eat up any unmatched character */
```

(The first three rules share the fourth's action since they use the special '|' action.) **REJECT** is a particularly expensive feature in terms scanner performance; if it is used in *any* of the scanner's actions it will slow down *all* of the scanner's matching. Furthermore, **REJECT**

cannot be used with the **–Cf** or **–CF** options (see below).

Note also that unlike the other special actions, **REJECT** is a *branch*; code immediately following it in the action will *not* be executed.

yymore( )
tells the scanner that the next time it matches a rule, the corresponding token should be *appended* onto the current value of *yytext* rather than replacing it. For example, given the input "mega-kludge" the following will write "mega-mega-kludge" to the output:

```
%%
mega-    ECHO; yymore();
kludge   ECHO;
```

First "mega-" is matched and echoed to the output. Then "kludge" is matched, but the previous "mega-" is still hanging around at the beginning of *yytext* so the **ECHO** for the "kludge" rule will actually write "mega-kludge". The presence of *yymore*( ) in the scanner's action entails a minor performance penalty in the scanner's matching speed.

yyless(*n*)
returns all but the first *n* characters of the current token back to the input stream, where they will be rescanned when the scanner looks for the next match. *yytext* and *yyleng* are adjusted appropriately (e.g., *yyleng* will now be equal to *n*). For example, on the input "foobar" the following will write out "foobarbar":

```
%%
foobar    ECHO; yyless(3);
[a-z]+    ECHO;
```

An argument of 0 to *yyless*( ) will cause the entire current input string to be scanned again. Unless you've changed how the scanner will subsequently process its input (using **BEGIN**, for example), this will result in an endless loop.

Note that **yyless** is a macro and can only be used in the lex input file, not from other source files.

unput(*c*)
puts the character *c* back onto the input stream. It will be the next character scanned. The following action will take the current token and cause it to be rescanned enclosed in parentheses.

```
{
int i;
unput( ')' );
for ( i = yyleng - 1; i >= 0; --i )
    unput( yytext[i] );
unput( '(' );
}
```

Note that since each *unput*( ) puts the given character back at the *beginning* of the input stream, pushing back strings must be done back-to-front. Also note that you cannot put back **EOF** to attempt to mark the input stream with an end-of-file.

input( ) reads the next character from the input stream. For example, the

following is one way to eat up C comments:

```
%%
"/*"        {
            register int c;

            for ( ; ; )
                {
                while ( (c = input()) != '*' &&
                        c != EOF )
                    ;    /* eat up text of comment */

                if ( c == '*' )
                    {
                    while ( (c = input()) == '*' )
                        ;
                    if ( c == '/' )
                        break;    /* found the end */
                    }

                if ( c == EOF )
                    {
                    error( "EOF in comment" );
                    break;
                    }
                }
            }
```

(Note that if the scanner is compiled using **C++**, then *input*( ) is instead referred to as *yyinput*( ), in order to avoid a name clash with the **C++** stream by the name of *input*.)

yyterminate()
can be used in lieu of a return statement in an action. It terminates the scanner and returns a 0 to the scanner's caller, indicating "all done". By default, *yyterminate*( ) is also called when an end-of-file is encountered. It is a macro and may be redefined.

THE GENERATED SCANNER

The output of *lex* is the file **lex.yy.c**, which contains the scanning routine *yylex*( ), a number of tables used by it for matching tokens, and a number of auxiliary routines and macros. By default, *yylex*( ) is declared as follows:

```
int yylex()
    {
    ... various definitions and the actions in here ...
    }
```

(If your environment supports function prototypes, then it will be "int yylex(void)".) This definition may be changed by defining the "YY_DECL" macro. For example, you could use:

```
#define YY_DECL float lexscan( a, b ) float a, b;
```

to give the scanning routine the name *lexscan*, returning a float, and taking two floats as arguments. Note that if you give arguments to the scanning routine using a K&R-style / non-prototyped function declaration, you must terminate

the definition with a semi-colon (;).

Whenever *yylex*( ) is called, it scans tokens from the global input file *yyin* (which defaults to stdin). It continues until it either reaches an end-of-file (at which point it returns the value 0) or one of its actions executes a *return* statement.

If the scanner reaches an end-of-file, subsequent calls are undefined unless either *yyin* is pointed at a new input file (in which case scanning continues from that file), or *yyrestart*( ) is called. *yyrestart*( ) takes one argument, a **FILE *** pointer, and initializes *yyin* for scanning from that file. Essentially there is no difference between just assigning *yyin* to a new input file or using *yyrestart*( ) to do so; the latter is available for compatibility with previous versions of *lex*, and because it can be used to switch input files in the middle of scanning. It can also be used to throw away the current input buffer, by calling it with an argument of *yyin*.

If *yylex*( ) stops scanning due to executing a *return* statement in one of the actions, the scanner may then be called again and it will resume scanning where it left off.

By default (and for purposes of efficiency), the scanner uses block-reads rather than simple *getc*( ) calls to read characters from *yyin*. The nature of how it gets its input can be controlled by defining the YY_INPUT macro. YY_INPUT's calling sequence is "YY_INPUT(buf,result,max_size)". Its action is to place up to *max_size* characters in the character array *buf* and return in the integer variable *result* either the number of characters read or the constant YY_NULL (traditionally 0) to indicate EOF. The default YY_INPUT reads from the global file-pointer "yyin".

A sample definition of YY_INPUT (in the definitions section of the input file):

```
%{
#define YY_INPUT(buf,result,max_size) \
    { \
    int c = getchar(); \
    result = (c == EOF) ? YY_NULL : (buf[0] = c, 1); \
    }
%}
```

This definition will change the input processing to occur one character at a time.

You also can add in things like keeping track of the input line number this way; but don't expect your scanner to go very fast.

When the scanner receives an end-of-file indication from YY_INPUT, it then checks the *yywrap*( ) function. If *yywrap*( ) returns false (zero), then it is assumed that the function has gone ahead and set up *yyin* to point to another input file, and scanning continues. If it returns true (non-zero), then the scanner terminates, returning 0 to its caller.

The default *yywrap*( ) always returns 1.

The scanner writes its **ECHO** output to the *yyout* global (default, stdout), which may be redefined by the user simply by assigning it to some other **FILE** pointer.

START CONDITIONS

lex provides a mechanism for conditionally activating rules. Any rule whose pattern is prefixed with "<sc>" will only be active when the scanner is in the start condition named "sc". For example,

```
<STRING>[^"]*        { /* eat up the string body ... */
             ...
             }
```

will be active only when the scanner is in the "STRING" start condition, and

```
<INITIAL,STRING,QUOTE>\.        { /* handle an escape ... */
             ...
             }
```

will be active only when the current start condition is either "INITIAL", "STRING", or "QUOTE".

Start conditions are declared in the definitions (first) section of the input using unindented lines beginning with either **%s** or **%x** followed by a list of names. The former declares *inclusive* start conditions, the latter *exclusive* start conditions. A start condition is activated using the **BEGIN** action. Until the next **BEGIN** action is executed, rules with the given start condition will be active and rules with other start conditions will be inactive. If the start condition is *inclusive*, then rules with no start conditions at all will also be active. If it is *exclusive*, then *only* rules qualified with the start condition will be active. A set of rules contingent on the same exclusive start condition describe a scanner which is independent of any of the other rules in the *lex* input. Because of this, exclusive start conditions make it easy to specify "mini-scanners" which scan portions of the input that are syntactically different from the rest (e.g., comments).

If the distinction between inclusive and exclusive start conditions is still a little vague, here's a simple example illustrating the connection between the two. The set of rules:

```
%s example
%%
<example>foo           /* do something */
```

is equivalent to

```
%x example
%%
<INITIAL,example>foo   /* do something */
```

Also note that the special start-condition specifier **<*>** matches every start condition. Thus, the above example could also have been written;

```
%x example
%%
<*>foo   /* do something */
```

The default rule (to **ECHO** any unmatched character) remains active in start conditions.

BEGIN(0) returns to the original state where only the rules with no start conditions are active. This state can also be referred to as the start-condition "INITIAL", so **BEGIN(INITIAL)** is equivalent to **BEGIN(0).** (The parentheses around the start condition name are not required but are considered good style.)

BEGIN actions can also be given as indented code at the beginning of the rules section. For example, the following will cause the scanner to enter the "SPECIAL" start condition whenever *yylex*( ) is called and the global variable *enter_special* is true:

```
        int enter_special;

%x SPECIAL
%%
        if ( enter_special )
            BEGIN(SPECIAL);

<SPECIAL>blahblahblah
...more rules follow...
```

To illustrate the uses of start conditions, here is a scanner which provides two different interpretations of a string like "123.456". By default it will treat it as as three tokens, the integer "123", a dot ('.'), and the integer "456". But if the string is preceded earlier in the line by the string "expect-floats" it will treat it as a single token, the floating-point number 123.456:

```
%{
#include <math.h>
%}
%s expect

%%
expect-floats        BEGIN(expect);

<expect>[0-9]+"."[0-9]+      {
            printf( "found a float, = %f\n",
                    atof( yytext ) );
            }
<expect>\n           {
            /* that's the end of the line, so
             * we need another "expect-number"
             * before we'll recognize any more
             * numbers
             */
            BEGIN(INITIAL);
            }
[0-9]+      {
            printf( "found an integer, = %d\n",
                    atoi( yytext ) );
            }

"."         printf( "found a dot\n" );
```

Here is a scanner which recognizes (and discards) C comments while maintaining a count of the current input line.

```
%x comment
%%
        int line_num = 1;

"/*"         BEGIN(comment);

<comment>[^*\n]*        /* eat anything that's not a '*' */
<comment>"*"+[^*/\n]*   /* eat up '*'s not followed by '/'s */
<comment>\n             ++line_num;
<comment>"*"+"/"        BEGIN(INITIAL);
```

This scanner goes to a bit of trouble to match as much text as possible with each rule. In general, when attempting to write a high-speed scanner try to match as much possible in each rule, as it's a big win.

Note that start-conditions names are really integer values and can be stored as such. Thus, the above could be extended in the following fashion:

```
%x comment foo
%%
        int line_num = 1;
        int comment_caller;

"/*"         {
             comment_caller = INITIAL;
             BEGIN(comment);
             }

...

<foo>"/*"    {
             comment_caller = foo;
             BEGIN(comment);
             }

<comment>[^*\n]*        /* eat anything that's not a '*' */
<comment>"*"+[^*/\n]*   /* eat up '*'s not followed by '/'s */
<comment>\n             ++line_num;
<comment>"*"+"/"        BEGIN(comment_caller);
```

Furthermore, you can access the current start condition using the integer-valued YY_START macro. For example, the above assignments to `comment_caller` could instead be written

```
comment_caller = YY_START;
```

Note that start conditions do not have their own name-space; %s's and %x's declare names in the same fashion as **#define**'s.

Finally, here's an example of how to match C-style quoted strings using exclusive start conditions, including expanded escape sequences (but not

including checking for a string that's too long):

```
%x str

%%
        char string_buf[MAX_STR_CONST];
        char *string_buf_ptr;

\"      string_buf_ptr = string_buf; BEGIN(str);

<str>\"        { /* saw closing quote - all done */
        BEGIN(INITIAL);
        *string_buf_ptr = '\0';
        /* return string constant token type and
         * value to parser
         */
        }

<str>\n        {
        /* error - unterminated string constant */
        /* generate error message */
        }

<str>\\[0-7]{1,3} {
        /* octal escape sequence */
        int result;

        (void) sscanf( yytext + 1, "%o", &result );

        if ( result > 0xff )
                /* error, constant is out-of-bounds */

        *string_buf_ptr++ = result;
        }

<str>\\[0-9]+ {
        /* generate error - bad escape sequence; something
         * like '\48' or '\0777777'
         */
        }

<str>\\n  *string_buf_ptr++ = '\n';
<str>\\t  *string_buf_ptr++ = '\t';
<str>\\r  *string_buf_ptr++ = '\r';
<str>\\b  *string_buf_ptr++ = '\b';
<str>\\f  *string_buf_ptr++ = '\f';

<str>\\(.|\n)  *string_buf_ptr++ = yytext[1];

<str>[^\\\n\"]+        {
        char *yytext_ptr = yytext;

        while ( *yytext_ptr )
                *string_buf_ptr++ = *yytext_ptr++;
        }
```

MULTIPLE INPUT BUFFERS

Some scanners (such as those which support "include" files) require reading from several input streams. As *lex* scanners do a large amount of buffering, one cannot control where the next input will be read from by simply writing a YY_INPUT which is sensitive to the scanning context. YY_INPUT is only called when the scanner reaches the end of its buffer, which may be a long time after scanning a statement such as an "include" which requires switching the input source.

To negotiate these sorts of problems, *lex* provides a mechanism for creating and switching between multiple input buffers. An input buffer is created by using:

```
YY_BUFFER_STATE yy_create_buffer( FILE *file, int size )
```

which takes a **FILE** pointer and a size and creates a buffer associated with the given file and large enough to hold *size* characters (when in doubt, use YY_BUF_SIZE for the size). It returns a YY_BUFFER_STATE handle, which may then be passed to other routines:

```
void yy_switch_to_buffer( YY_BUFFER_STATE new_buffer )
```

switches the scanner's input buffer so subsequent tokens will come from *new_buffer*. Note that *yy_switch_to_buffer*( ) may be used by *yywrap*( ) to set things up for continued scanning, instead of opening a new file and pointing *yyin* at it.

```
void yy_delete_buffer( YY_BUFFER_STATE buffer )
```

is used to reclaim the storage associated with a buffer.

yy_new_buffer( ) is an alias for *yy_create_buffer*( ), provided for compatibility with the C++ use of *new* and *delete* for creating and destroying dynamic objects.

Finally, the YY_CURRENT_BUFFER macro returns a YY_BUFFER_STATE handle to the current buffer.

Here is an example of using these features for writing a scanner which expands include files (the **<<EOF>>** feature is discussed below):

```
/* the "incl" state is used for picking up the name
 * of an include file
 */
%x incl

%{
#define MAX_INCLUDE_DEPTH 10
YY_BUFFER_STATE include_stack[MAX_INCLUDE_DEPTH];
int include_stack_ptr = 0;
%}

%%
include             BEGIN(incl);

[a-z]+              ECHO;
[^a-z\n]*\n?        ECHO;

<incl>[ \t]*      /* eat the whitespace */
```

```
<incl>[^ \t\n]+   { /* got the include file name */
        if ( include_stack_ptr >= MAX_INCLUDE_DEPTH )
            {
            fprintf( stderr, "Includes nested too deeply" );
            exit( 1 );
            }

        include_stack[include_stack_ptr++] =
            YY_CURRENT_BUFFER;

        yyin = fopen( yytext, "r" );

        if ( ! yyin )
            error( ... );

        yy_switch_to_buffer(
            yy_create_buffer( yyin, YY_BUF_SIZE ) );

        BEGIN(INITIAL);
        }

<<EOF>> {
        if ( --include_stack_ptr < 0 )
            {
            yyterminate();
            }

        else
            {
            yy_delete_buffer( YY_CURRENT_BUFFER );
            yy_switch_to_buffer(
                 include_stack[include_stack_ptr] );
            }
        }
```

END-OF-FILE RULES

The special rule "<<EOF>>" indicates actions which are to be taken when an end-of-file is encountered and *yywrap*( ) returns non-zero (i.e., indicates no further files to process). The action must finish by doing one of four things:

- assigning *yyin* to a new input file (in previous versions of lex, after doing the assignment you had to call the special action YY_NEW_FILE; this is no longer necessary);
- executing a **return** statement;
- executing the special *yyterminate*( ) action;
- or, switching to a new buffer using *yy_switch_to_buffer*( ) as shown in the example above.

<<EOF>> rules may not be used with other patterns; they may only be qualified with a list of start conditions. If an unqualified <<EOF>> rule is given, it applies to *all* start conditions which do not already have <<EOF>> actions. To specify an <<EOF>> rule for only the initial start condition, use

```
<INITIAL><<EOF>>
```

These rules are useful for catching things like unclosed comments. An example:

```
%x quote
%%

...other rules for dealing with quotes...

<quote><<EOF>>   {
         error( "unterminated quote" );
         yyterminate();
         }
<<EOF>>  {
         if ( *++filelist )
             yyin = fopen( *filelist, "r" );
         else
            yyterminate();
         }
```

MISCELLANEOUS MACROS

The macro YY_USER_ACTION can be defined to provide an action which is always executed prior to the matched rule's action. For example, it could be **#define**'d to call a routine to convert *yytext* to lower-case.

The macro YY_USER_INIT may be defined to provide an action which is always executed before the first scan (and before the scanner's internal initializations are done). For example, it could be used to call a routine to read in a data table or open a logging file.

In the generated scanner, the actions are all gathered in one large switch statement and separated using YY_BREAK, which may be redefined. By default, it is simply a "break", to separate each rule's action from the following rule's. Redefining YY_BREAK allows, for example, C++ users to **#define** YY_BREAK to do nothing (while being very careful that every rule ends with a "break" or a "return"!) to avoid suffering from unreachable statement warnings where because a rule's action ends with "return", the YY_BREAK is inaccessible.

INTERFACING WITH YACC

One of the main uses of *lex* is as a companion to the *yacc* parser-generator. *yacc* parsers expect to call a routine named *yylex*() to find the next input token. The routine is supposed to return the type of the next token as well as putting any associated value in the global *yylval*. To use *lex* with *yacc*, one specifies the **–d** option to *yacc* to instruct it to generate the file **y.tab.h** containing definitions of all the **%tokens** appearing in the *yacc* input. This file is then included in the *lex* scanner. For example, if one of the tokens is "TOK_NUMBER", part of the scanner might look like:

```
%{
#include "y.tab.h"
%}

%%
```

```
[0-9]+        yylval = atoi( yytext ); return TOK_NUMBER;
```

LEX OPTIONS

lex has the following options:

–7 Generate a 7-bit scanner, which can save considerable table space, especially when using **–Cf** or **–CF** (and, at most sites, **–7** is on by default for these options. To see if this is the case, use the **–v** verbose flag and check the flag summary it reports).

–8 Generate an 8-bit scanner. This is the default except for the **–Cf** and **–CF** compression options, for which the default is site-dependent, and can be checked by inspecting the flag summary generated by the **–v** option.

–B Generate a *batch* scanner instead of an *interactive* scanner (see **–I** below). See *lexdoc* for details. Scanners using **–Cf** or **–CF** compression options automatically specify this option, too.

–b Generate backing-up information to *lex.backup*. This is a list of scanner states which require backing up and the input characters on which they do so. By adding rules one can remove backing-up states. If all backing-up states are eliminated and **–Cf** or **–CF** is used, the generated scanner will run faster.

–C[aefFmr] Specify degree of table compression and scanner optimization.

–Ca Trade off larger tables in the generated scanner for faster performance because the elements of the tables are better aligned for memory access and computation. This option can double the size of the tables used by your scanner.

–Ce Construct *equivalence* classes, i.e., sets of characters which have identical lexical properties. Equivalence classes usually give dramatic reductions in the final table / object file sizes (typically a factor of 2-5) and are pretty cheap performance-wise (one array look-up per character scanned).

–Cf Generate *full* scanner tables — *lex* should not compress the tables by taking advantages of similar transition functions for different states.

–CF Use the alternate fast scanner representation (described in *lexdoc*).

–Cm Construct *meta-equivalence classes*, which are sets of equivalence classes (or characters, if equivalence classes are not being used) that are commonly used together. Meta-equivalence classes are often a big win when using compressed tables, but they have a moderate performance impact (one or two "if" tests and one array look-up per character scanned).

–Cr *Bypass* using stdio for input in generated scanner. In general this option results in a minor performance gain only worthwhile if used in conjunction with **–Cf** or **–CF**. It can cause surprising behavior if you use stdio yourself to read

from **yyin** prior to calling the scanner.

–C Alone, compress scanner tables but use neither equivalence classes nor meta-equivalence classes.

The options **–Cf** or **–CF** and **–Cm** do not make sense together — there is no opportunity for meta-equivalence classes if the table is not being compressed. Otherwise the options may be freely mixed.

The default setting is **–Cem**, which specifies that *lex* should generate equivalence classes and meta-equivalence classes. This setting provides the highest degree of table compression. You can trade off faster-executing scanners at the cost of larger tables with the following generally being true:

slowest & smallest	**–Cem**
	–Cm
	–Ce
	–C
	–C{f,F}e
	–C{f,F}
fastest & largest	**–C{f,F}a**

–C options are cumulative.

–c Does nothing; a deprecated option included for POSIX compliance.

NOTE: In previous releases of *lex* **–c** specified table-compression options. This functionality is now given by the **–C** flag. To ease the the impact of this change, when *lex* encounters **–c**, it currently issues a warning message and assumes that **–C** was desired instead. In the future this "promotion" of **–c** to **–C** will go away in the name of full POSIX compliance (unless the POSIX meaning is removed first).

–d Run the generated scanner in *debug* mode. Whenever a pattern is recognized and the global *yy_flex_debug* is non-zero (which is the default), the scanner will write to stderr a line of the form:

```
--accepting rule at line 53 ("the matched text")
```

The line number refers to the location of the rule in the file defining the scanner (i.e., the file that was fed to lex). Messages are also generated when the scanner backs up, accepts the default rule, reaches the end of its input buffer (or encounters a NUL; the two look the same as far as the scanner's concerned), or reaches an end-of-file.

–F Use *fast* scanner table representation (and bypass stdio). This representation is about as fast as the full table representation (**–f)**, and for some sets of patterns will be considerably smaller (and for others, larger). See *lexdoc* for more details.

This option is equivalent to **–CFr** (see below).

–f Use *fast scanner*. No table compression is done and stdio is bypassed. The result is large but fast. This option is equivalent to **–Cfr** (see below).

–h Generate a "help" summary of *lex's* options to stderr and then exits.

–I Generate an *interactive* scanner, that is, a scanner which stops immediately rather than looking ahead if it knows that the currently scanned text cannot be part of a longer rule's match. This is the opposite of *batch* scanners (see **–B** above). See *lexdoc* for details.

Note, **–I** cannot be used in conjunction with *full* or *fast tables*, i.e., the **–f**, **–F**, **–Cf**, or **–CF** flags. For other table compression options, **–I** is the default.

–i Generate a *case-insensitive* scanner. The case of letters given in the *lex* input patterns will be ignored, and tokens in the input will be matched regardless of case. The matched text given in *yytext* will have the preserved case (i.e., it will not be folded).

–L Do *note* generate **#line** directives in **lex.yy.c**. The default is to generate such directives so error messages in the actions will be correctly located with respect to the original *lex* input file, and not to the fairly meaningless line numbers of **lex.yy.c**.

–l Turn on maximum compatibility with the original AT&T lex implementation, at a considerable performance cost. This option is incompatible with **–f**, **–F**, **–Cf**, or **–CF.** See *lexdoc* for details.

–n Does nothing. Another deprecated option included only for POSIX compliance.

–P*prefix*

Change the default *yy* prefix used by *lex* to be *prefix* instead. See *lexdoc* for a description of all the global variables and file names that this affects.

–p Generate a performance report to stderr. The report consists of comments regarding features of the *lex* input file which will cause a loss of performance in the resulting scanner. If you give the flag twice, you will also get comments regarding features that lead to minor performance losses.

–S*skeleton_file*

Use *skeleton_file* to construct the scanner instead of the default file. You'll never need this option unless you are doing *lex* maintenance or development.

–s Suppress the *default rule* (that unmatched scanner input is echoed to stdout). If the scanner encounters input that does not match any of its rules, it aborts with an error.

–T Run in *trace* mode. It will generate a lot of messages to stderr concerning the form of the input and the resultant non-deterministic and deterministic finite automata. This option is mostly for use in maintaining *lex*.

–t Write the scanner it generates to standard output instead of **lex.yy.c**.

–V Print the version number to stderr and exits.

–v Write to stderr a summary of statistics regarding the scanner it generates.

–w Suppress warning messages.

PERFORMANCE CONSIDERATIONS

The main design goal of *lex* is that it generate high-performance scanners. It has been optimized for dealing well with large sets of rules. Aside from the effects on scanner speed of the table compression **–C** options outlined above, there are a number of options / actions which degrade performance. These are, from most expensive to least:

- **REJECT**
- pattern sets that require backing up
- arbitrary trailing context
- *yymore*( )
- ^ beginning-of-line operator

with the first three all being quite expensive and the last two being quite cheap. Note also that *unput*( ) is implemented as a routine call that potentially does quite a bit of work, while *yyless*( ) is a quite-cheap macro; so if just putting back some excess text you scanned, use *yyless*( ).

Avoid **REJECT** at all costs when performance is important. It is a particularly expensive option.

Getting rid of backing up is messy and often may be an enormous amount of work for a complicated scanner. In principal, one begins by using the **–b** flag to generate a **lex.backup** file. For example, on the input

```
%%
foo        return TOK_KEYWORD;
foobar     return TOK_KEYWORD;
```

the file looks like:

```
State #6 is non-accepting -
 associated rule line numbers:
       2       3
 out-transitions: [ o ]
 jam-transitions: EOF [ \001-n  p-\177 ]

State #8 is non-accepting -
 associated rule line numbers:
       3
 out-transitions: [ a ]
 jam-transitions: EOF [ \001-‘  b-\177 ]

State #9 is non-accepting -
 associated rule line numbers:
       3
```

```
 out-transitions: [ r ]
 jam-transitions: EOF [ \001-q  s-\177 ]

Compressed tables always back up.
```

The first few lines tell us that there's a scanner state in which it can make a transition on an 'o' but not on any other character, and that in that state the currently scanned text does not match any rule. The state occurs when trying to match the rules found at lines 2 and 3 in the input file. If the scanner is in that state and then reads something other than an 'o', it will have to back up to find a rule which is matched. With a bit of concentrated thought, one can see that this must be the state the scanner is in when it has seen "fo". When this has happened, if anything other than another 'o' is seen, the scanner will have to back up to simply match the 'f' (by the default rule).

The comment regarding State #8 indicates there's a problem when "foob" has been scanned. Indeed, on any character other than an 'a', the scanner will have to back up to accept "foo". Similarly, the comment for State #9 concerns when "fooba" has been scanned and an 'r' does not follow.

The final comment reminds us that there's no point going to all the trouble of removing backing up from the rules unless we're using **–Cf** or **–CF**, since there's no performance gain doing so with compressed scanners.

The way to remove the backing up is to add "error" rules:

```
%%
foo         return TOK_KEYWORD;
foobar      return TOK_KEYWORD;

fooba       |
foob        |
fo          {
            /* false alarm, not really a keyword */
            return TOK_ID;
            }
```

Eliminating backing up among a list of keywords can also be done using a "catch-all" rule:

```
%%
foo         return TOK_KEYWORD;
foobar      return TOK_KEYWORD;

[a-z]+      return TOK_ID;
```

This is usually the best solution when appropriate.

Backing up messages tend to cascade. With a complicated set of rules it's not uncommon to get hundreds of messages. If one can decipher them, though, it often only takes a dozen or so rules to eliminate the backing up (though it's easy to make a mistake and have an error rule accidentally match a valid token. A possible future *lex* feature will be to automatically add rules to eliminate backing up).

Variable trailing context (where both the leading and trailing parts do not have a fixed length) entails almost the same performance loss as **REJECT** (i.e., substantial). So when possible a rule like:

```
%%
mouse|rat/(cat|dog)   run();
```

is better written:

```
%%
mouse/cat|dog         run();
rat/cat|dog           run();
```

or as

```
%%
mouse|rat/cat         run();
mouse|rat/dog         run();
```

Note that here the special '|' action does *not* provide any savings, and can even make things worse (see

A final note regarding performance: as mentioned above in the section How the Input is Matched, dynamically resizing *yytext* to accommodate huge tokens is a slow process because it presently requires that the (huge) token be rescanned from the beginning. Thus if performance is vital, you should attempt to match "large" quantities of text but not "huge" quantities, where the cutoff between the two is at about 8K characters/token.

Another area where the user can increase a scanner's performance (and one that's easier to implement) arises from the fact that the longer the tokens matched, the faster the scanner will run. This is because with long tokens the processing of most input characters takes place in the (short) inner scanning loop, and does not often have to go through the additional work of setting up the scanning environment (e.g., *yytext*) for the action. Recall the scanner for C comments:

```
%x comment
%%
        int line_num = 1;

"/*"         BEGIN(comment);

<comment>[^*\n]*
<comment>"*"+[^*/\n]*
<comment>\n             ++line_num;
<comment>"*"+"/"        BEGIN(INITIAL);
```

This could be sped up by writing it as:

```
%x comment
%%
        int line_num = 1;

"/*"         BEGIN(comment);

<comment>[^*\n]*
```

```
<comment>[^*\n]*\n      ++line_num;
<comment>"*"+[^*/\n]*
<comment>"*"+[^*/\n]*\n ++line_num;
<comment>"*"+"/"        BEGIN(INITIAL);
```

Now instead of each newline requiring the processing of another action, recognizing the newlines is "distributed" over the other rules to keep the matched text as long as possible. Note that *adding* rules does *not* slow down the scanner! The speed of the scanner is independent of the number of rules or (modulo the considerations given at the beginning of this section) how complicated the rules are with regard to operators such as '*' and '|'.

A final example in speeding up a scanner: suppose you want to scan through a file containing identifiers and keywords, one per line and with no other extraneous characters, and recognize all the keywords. A natural first approach is:

```
%%
asm      |
auto     |
break    |
... etc ...
volatile |
while    /* it's a keyword */

.|\n     /* it's not a keyword */
```

To eliminate the back-tracking, introduce a catch-all rule:

```
%%
asm      |
auto     |
break    |
... etc ...
volatile |
while    /* it's a keyword */

[a-z]+   |
.|\n     /* it's not a keyword */
```

Now, if it's guaranteed that there's exactly one word per line, then we can reduce the total number of matches by a half by merging in the recognition of newlines with that of the other tokens:

```
%%
asm\n    |
auto\n   |
break\n  |
... etc ...
volatile\n |
while\n  /* it's a keyword */

[a-z]+\n |
.|\n     /* it's not a keyword */
```

One has to be careful here, as we have now reintroduced backing up into the scanner. In particular, while *we* know that there will never be any characters in the input stream other than letters or newlines, *lex* can't figure this out, and it will plan for possibly needing to back up when it has scanned a token like "auto" and then the next character is something other than a newline or a letter. Previously it would then just match the "auto" rule and be done, but now it has no "auto" rule, only a "auto\n" rule. To eliminate the possibility of backing up, we could either duplicate all rules but without final newlines, or, since we never expect to encounter such an input and therefore don't how it's classified, we can introduce one more catch-all rule, this one which doesn't include a newline:

```
%%
asm\n    |
auto\n   |
break\n  |
... etc ...
volatile\n |
while\n  /* it's a keyword */

[a-z]+\n |
[a-z]+   |
.|\n     /* it's not a keyword */
```

Compiled with **–Cf**, this is about as fast as one can get a *lex* scanner to go for this particular problem.

A final note: *lex* is slow when matching NUL's, particularly when a token contains multiple NUL's. It's best to write rules which match *short* amounts of text if it's anticipated that the text will often include NUL's.

INCOMPATIBILITIES WITH AT&T LEX AND POSIX

This *lex* is a rewrite of the AT&T *lex* tool (the two implementations do not share any code, though), with some extensions and incompatibilities, both of which are of concern to those who wish to write scanners acceptable to either implementation. The POSIX *lex* specification is closer to this *lex's* behavior than that of the original AT&T *lex* implementation, but there also remain some incompatibilities between this *lex* and POSIX. The intent is that ultimately this *lex* will be fully POSIX-conformant. In this section we discuss all of the known areas of incompatibility.

lex's **–l** option turns on maximum compatibility with the original AT&T *lex* implementation, at the cost of a major loss in the generated scanner's performance. We note below which incompatibilities can be overcome using the **–l** option.

lex is fully compatible with AT&T *lex* with the following exceptions:

- The undocumented AT&T *lex* scanner internal variable *yylineno* is not supported unless **–l** is used.
- *yylineno* is not part of the POSIX specification.
- The *input*( ) routine is not redefinable, though it may be called to read characters following whatever has been matched by a rule. If *input*( ) encounters an end-of-file the normal *yywrap*( ) processing is done. A ''real''

end-of-file is returned by *input*( ) as EOF.

- Input is instead controlled by defining the YY_INPUT macro.
- The *lex* restriction that *input*( ) cannot be redefined is in accordance with the POSIX specification, which simply does not specify any way of controlling the scanner's input other than by making an initial assignment to *yyin*.
- These *lex* scanners are not as reentrant as AT&T *lex* scanners. In particular, if you have an interactive scanner and an interrupt handler which long-jumps out of the scanner, and the scanner is subsequently called again, you may get the following message:

  fatal lex scanner internal error--end of buffer missed

  To reenter the scanner, first use

  ```
  yyrestart( yyin );
  ```

  Note that this call will throw away any buffered input; usually this isn't a problem with an interactive scanner.
- *Output*( ) is not supported. Output from the **ECHO** macro is done to the file-pointer *yyout* (default stdout).
- *output*( ) is not part of the POSIX specification.
- AT&T *lex* does not support exclusive start conditions (%x), though they are in the POSIX specification.
- When definitions are expanded, *lex* encloses them in parentheses. With AT&T lex, the following:

  ```
  NAME    [A-Z][A-Z0-9]*
  %%
  foo{NAME}?      printf( "Found it\n" );
  %%
  ```

  will not match the string "foo" because when the macro is expanded the rule is equivalent to "foo[A-Z][A-Z0-9]*?" and the precedence is such that the '?' is associated with "[A-Z0-9]*". With this *lex*, the rule will be expanded to "foo([A-Z][A-Z0-9]*)?" and so the string "foo" will match.
- Note that if the definition begins with ^ or ends with **$** then it is *not* expanded with parentheses, to allow these operators to appear in definitions without losing their special meanings. But the **<s>**, **/**, and **<<EOF>>** operators cannot be used in a *lex* definition.
- Using **–l** results in the AT&T *lex* behavior of no parentheses around the definition.
- The POSIX specification is that the definition be enclosed in parentheses.
- The AT&T *lex* **%r** (generate a Ratfor scanner) option is not supported. It is not part of the POSIX specification. After a call to *unput*( ), *yytext* and *yyleng* are undefined until the next token is matched, unless the scanner was built using **%array**. This is not the case with AT&T *lex* or the POSIX specification. The **–l** option does away with this incompatibility.

- The precedence of the **{}** (numeric range) operator is different. AT&T *lex* interprets "abc{1,3}" as "match one, two, or three occurrences of 'abc'", whereas *lex* interprets it as "match 'ab' followed by one, two, or three occurrences of 'c'". The latter is in agreement with the POSIX specification.
- The precedence of the ^ operator is different. AT&T *lex* interprets "^foo|bar" as "match either 'foo' at the beginning of a line, or 'bar' anywhere", whereas *lex* interprets it as "match either 'foo' or 'bar' if they come at the beginning of a line". The latter is in agreement with the POSIX specification.
- *yyin* is *initialized* by AT&T *lex* to be stdin; *lex*, on the other hand, initializes *yyin* to NULL and then *assigns* it to *stdin* the first time the scanner is called, providing *yyin* has not already been assigned to a non-NULL value. The difference is subtle, but the net effect is that with *lex* scanners, *yyin* does not have a valid value until the scanner has been called.

  The **–l** option does away with this incompatibility.
- The special table-size declarations such as **%a** supported by AT&T and POSIX *lex* are not required by *lex* scanners; *lex* ignores them.
- The name FLEX_SCANNER is **#define**'d so scanners may be written for use with either this *lex* (which is actually a version of the *flex* scanner-generator) or AT&T *lex*.

The following *lex* features are not included in AT&T *lex* or the POSIX specification:

— *yyterminate*( )

— <<EOF>>

— <*>

— YY_DECL

— YY_START

— YY_USER_ACTION

— #line directives

— %{}'s around actions

— multiple actions on a line

plus almost all of the *lex* flags. The last feature in the list refers to the fact that with *lex* you can put multiple actions on the same line, separated with semi-colons, while with AT&T *lex*, the following

```
foo     handle_foo(); ++num_foos_seen;
```

is (rather surprisingly) truncated to

```
foo     handle_foo();
```

lex does not truncate the action. Actions that are not enclosed in braces are simply terminated at the end of the line.

DIAGNOSTICS

Lex produces the following diagnostic messages:

warning, rule cannot be matched

Indicates that the given rule cannot be matched because it follows other rules that will always match the same text as it. For example, in the following "foo" cannot be matched because it comes after an identifier "catch-all" rule:

```
[a-z]+    got_identifier();
foo       got_foo();
```

Using **REJECT** in a scanner suppresses this warning.

warning, –s option given but default rule can be matched

It is possible (perhaps only in a particular start condition) that the default rule (match any single character) is the only one that will match a particular input. Since **–s** was given, presumably this is not intended.

reject_used_but_not_detected undefined

This is a compile-time error; the scanner uses **REJECT** but *lex* didn't find it in the first two sections. Usually this happens because the reference is implicit (via a #include file, for example). Make an explicit reference to the action in your *lex* input file. (Note that previously *lex* supported a **%used**/**%unused** mechanism for dealing with this problem; this feature is still supported but now deprecated, and will go away soon unless the author hears from people who can argue compellingly that they need it.)

yymore_used_but_not_detected undefined

This is a compile-time error; the scanner uses *yymore*( ) but *lex* didn't find it in the first two sections. Usually this happens because the reference is implicit (via a #include file, for example). Make an explicit reference to the action in your *lex* input file. (Note that previously *lex* supported a **%used**/**%unused** mechanism for dealing with this problem; this feature is still supported but now deprecated, and will go away soon unless the author hears from people who can argue compellingly that they need it.)

lex scanner jammed

A scanner compiled with **–s** has encountered an input string which wasn't matched by any of its rules. This error can also occur due to internal problems.

token too large, exceeds YYLMAX

Your scanner uses **%array** and one of its rules matched a string longer than the YYLMAX constant (8K bytes by default). You can increase the value by **#define**'ing YYLMAX in the definitions section of your *lex* input.

scanner requires –8 flag to use the character 'x'

Your scanner specification includes recognizing the 8-bit character *'x'* and you did not specify the **–8** flag, and your scanner defaulted to 7-bit because you used the **–Cf** or **–CF** table compression options. See the discussion of the **–7** flag for details.

lex scanner push-back overflow

You used *unput*( ) to push back so much text that the scanner's buffer could not hold both the pushed-back text and the current token in *yytext*. Ideally the scanner should dynamically resize the buffer in this case, but at present it does not.

input buffer overflow, can't enlarge buffer because scanner uses REJECT

The scanner was working on matching an extremely large token and needed to expand the input buffer. This doesn't work with scanners that use **REJECT**.

fatal lex scanner internal error--end of buffer missed

This can occur in an scanner which is reentered after a long-jump has jumped out (or over) the scanner's activation frame. Before reentering the scanner, use:

```
yyrestart( yyin );
```

too many start conditions in <> construct!

You listed more start conditions in a <> construct than exist (so you must have listed at least one of them twice).

FILES

See *lex*.

DEFICIENCIES / BUGS

Again, see *lex*.

AUTHOR

Vern Paxson, with the help of many ideas and much inspiration from Van Jacobson. Original version by Jef Poskanzer. The fast table representation is a partial implementation of a design done by Van Jacobson. The implementation was done by Kevin Gong and Vern Paxson.

Thanks to the many *lex* beta-testers, feedbackers, and contributors, especially Francois Pinard, Casey Leedom, Nelson H.F. Beebe, benson@odi.com, Peter A. Bigot, Keith Bostic, Frederic Brehm, Nick Christopher, Jason Coughlin, Bill Cox, Dave Curtis, Scott David Daniels, Chris G. Demetriou, Mike Donahue, Chuck Doucette, Tom Epperly, Leo Eskin, Chris Faylor, Jon Forrest, Kaveh R. Ghazi, Eric Goldman, Ulrich Grepel, Jan Hajic, Jarkko Hietaniemi, Eric Hughes, John Interrante, Ceriel Jacobs, Jeffrey R. Jones, Henry Juengst, Amir Katz, ken@ken.hilco.com, Kevin B. Kenny, Marq Kole, Ronald Lamprecht, Greg Lee, Craig Leres, John Levine, Steve Liddle, Mohamed el Lozy, Brian Madsen, Chris Metcalf, Luke Mewburn, Jim Meyering, G.T. Nicol, Landon Noll, Marc Nozell, Richard Ohnemus, Sven Panne, Roland Pesch, Walter Pelissero, Gaumond Pierre, Esmond Pitt, Jef Poskanzer, Joe Rahmeh, Frederic Raimbault, Rick Richardson, Kevin Rodgers, Jim Roskind, Doug Schmidt, Philippe Schnoebelen, Andreas Schwab, Alex Siegel, Mike Stump, Paul Stuart, Dave Tallman, Chris Thewalt, Paul Tuinenga, Gary Weik, Frank Whaley, Gerhard Wilhelms, Kent Williams, Ken Yap, Nathan Zelle, David Zuhn, and those whose names have slipped my marginal mail-archiving skills but whose contributions are appreciated all the same.

Thanks to Keith Bostic, Jon Forrest, Noah Friedman, John Gilmore, Craig Leres, John Levine, Bob Mulcahy, G.T. Nicol, Francois Pinard, Rich Salz, and Richard Stallman for help with various distribution headaches.

Thanks to Esmond Pitt and Earle Horton for 8-bit character support; to Benson Margulies and Fred Burke for C++ support; to Kent Williams and Tom Epperly for C++ class support; to Ove Ewerlid for support of NUL's; and to Eric Hughes for support of multiple buffers.

This work was primarily done when I was with the Real Time Systems Group at the Lawrence Berkeley Laboratory in Berkeley, CA. Many thanks to all there

for the support I received.

Send comments to:

Vern Paxson
Systems Engineering
Bldg. 46A, Room 1123
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

vern@ee.lbl.gov

SEE ALSO

lex, *yacc*, *sed*, *awk*.

M. E. Lesk and E. Schmidt, *LEX – Lexical Analyzer Generator*

NAME

line – read a line of text

SYNOPSIS

```
line
```

DESCRIPTION

The *line* utility reads one line of text from standard input and writes it to standard output. It reads up to and including a newline; and always writes at least a newline.

Historically, *line* is used in shell scripts to read from the user's terminal; the *read* command is preferred.

DIAGNOSTICS

The *line* command exits with status 0 on success and >0 if end-of-file is found.

SEE ALSO

read

NAME

listres - list resources in widgets

SYNOPSIS

`listres [-option ...]`

DESCRIPTION

The *listres* program generates a list of a widget's resource database. The class in which each resource is first defined, the instance and class name, and the type of each resource is listed. If no specific widgets or the **–all** switch are given, a two-column list of widget names and their class hierarchies is printed.

OPTIONS

Listres accepts all of the standard toolkit command line options along with those listed below:

–all This option indicates that *listres* should print information for all known widgets and objects.

–nosuper
This option indicates that resources that are inherited from a superclass should not be listed. This is useful for determining which resources are new to a subclass.

–variable
This option indicates that widgets should be identified by the names of the class record variables rather than the class name given in the variable. This is useful for distinguishing subclasses that have the same class name as their superclasses.

–top *name*
This option specifies the name of the widget to be treated as the top of the hierarchy. Case is not significant, and the name may match either the class variable name or the class name. The default is ''core''.

–format *printf–string*
This option specifies the *printf*-style format string to be used to print out the name, instance, class, and type of each resource.

SEE ALSO

X, *xrdb*, appropriate widget documents

BUGS

On operating systems that do not support dynamic linking of run-time routines, this program must have all of its known widgets compiled in. The sources provide several tools for automating this process for various widget sets.

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

ln – link files

SYNOPSIS

```
ln [-f] source_file target_file
ln [-f] source_file...  target_dir
```

DESCRIPTION

The *ln* utility creates a new directory entry (linked file) which has the same modes as the original file. It is useful for maintaining multiple copies of a file in many places at once without using up storage for the ‘‘copies’’; instead, a link ‘‘points’’ to the original copy.

The options are as follows:

–f Unlink any already existing file, permitting the link to occur.

The *ln* utility makes hard links. A hard link to a file is indistinguishable from the original directory entry; changes to a file take effect regardless of the name used to refer to the file. Hard links may not normally refer to directories and may not span file systems.

Given one or two arguments, *ln* creates a link to an existing file *source_file*. If *target_file* is given, the link has that name; *target_file* may also be a directory in which to place the link; otherwise it is placed in the current directory. If only the directory is specified, the link will be made to the last component of *source_file*.

Given more than two arguments, *ln* makes links in *target_dir* to all the named source files. The links made will have the same name as the files being linked to. One cannot hard link a directory.

DIAGNOSTICS

The *ln* utility exits 0 on success, and >0 if an error occurs.

NOTES

The *ln* command requires support from the underlying file system; it will not work on files on a FAT file system or on network mounted drives.

SEE ALSO

cp, *mv*, *rm*

NAME

locale – get locale-specific information

SYNOPSIS

```
locale [-ck] name ...
locale [-a] [-m]
```

DESCRIPTION

The *locale* utility writes information to standard-output about the current locale.

The options are as follows:

–a Write the names of all public available locales. Currently, only the *C* and *POSIX* locales are available.

–c Write the name of the selected locale categories.

–k Write the names and values of selected locale category keywords.

–m Write the names of all available character maps.

Refer to the POSIX.2 standard for a complete discussion of the Locale Categories and Keywords.

DIAGNOSTICS

The *locale* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

localedef

NAME

localedef – define locale-environment

SYNOPSIS

`localedef [-c] [-f charmap] [-i sourcefile] name`

DESCRIPTION

The *localedef* utility does not currently support the definition of user locales. It essentially parses its command-line arguments.

The options are as follows:

–c Create permanent output despite warning messages.

–f *charmap*
Specify the pathname of the character mapping file (*charmap*).

–i *sourcefile*
Specify the pathname of the source file for the locale definition. Otherwise, standard-input is read.

DIAGNOSTICS

The INTERIX *localedef* utility always exits with status 3 because user-defined locales are not supported. Possible status values for the utility are:

0 Success and the locale was successfully created.

1 Warnings (such as problems parsing the command line) but the locale was successfully created.

2 Errors—the specification exceeded implementation limits or the character set or sets used were not supported by the implementation, and no locale was created.

3 User-defined locales are not supported by this implementation.

>3 Error or warning and no locale was created.

SEE ALSO

locale

NAME

logger – log messages

SYNOPSIS

`logger` *`string`*`...`

DESCRIPTION

The *logger* utility writes its arguments (a message) to the log file **$OPENNT_ROOT/var/adm/log/logger**, where the value of *OPENNT_ROOT* is the installation directory for INTERIX. The date and time, and the user name are also logged with the message:

```
19Mar1996 21:31:42 EST (ralph): CSRR10001 Started.
19Mar1996 21:31:50 EST (ralph): CSRR10001 Finished.
20Mar1996 20:11:22 EST (tim): JOB960320.1 Changed access codes.
```

There are no options.

DIAGNOSTICS

The *logger* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

syslogd

NAME

login – log into the computer

SYNOPSIS

`login [-fIp] [-h hostname] [user] ...`

DESCRIPTION

The *login* utility logs users into the computer system.

If no *user* is specified, or if a *user* is specified and authentication of the user fails, *login* prompts for a user name. Authentication of users is done via passwords.

Login takes the following options:

–f Force login without a password; it indicates that the proper authentication has already been done and no password need be requested. This may only be used when an already logged-in user is logging in as him or herself.

–h *hostname*
Specifies that the connection was received from *hostname*. This option is used by various daemons, and requires appropriate privileges.

–I Before prompting for *user*, *login* prints information about the *domain+username* format. If a *user* is supplied on the command line, *login* prints this information only if the password is wrong.

–p Preserve the environment. By default, *login* discards the previous environment.

Login enters information into the environment specifying the user's home directory (*HOME*), command interpreter (*SHELL*), search path (*PATH*), terminal type (*TERM*) and user name (both *LOGNAME* and *USER*).

The standard shells, *csh* and *sh*, do not fork before executing the *login* utility.

NOTES

The default domain presented in the login banner can be changed by setting the Windows NT registry variable **HKEY_LOCAL_MACHINE** | **SOFTWARE** | **Softway Systems** | **OpenNT** | **LoginDefaultDomain** to the name of the domain.

On a Windows NT system running in a workgroup without a domain controller, users must type "*+loginname*". To avoid this, set the value of **LoginDefaultDomain** to an empty string.

NAME

logname – display user's login name

SYNOPSIS

`logname`

DESCRIPTION

The *logname* utility writes the user's login name to standard output followed by a newline.

The *logname* utility explicitly ignores the *LOGNAME* and *USER* environment variables because the environment cannot be trusted.

DIAGNOSTICS

The *logname* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

id, *who*

NAME

lp – send files to a printer

SYNOPSIS

`lp [-bc] [-d dest] [-n copies] [file...]`

DESCRIPTION

The *lp* utility sends files to a system printer, identified by **–d** *dest*.

The options are as follows:

–b Binary file; don't convert line-endings to Windows NT text format.

–c Copy the files so that the user can modify / delete *file* without affecting the operation.

–d *dest*

Names the output device to which the files will be written. The destination is specified in UNC format: / / SYSTEM / printer. For example, the printer "Scrivener" on the system "Bartleby" would be specified `//BARTLEBY/Scrivener`.

This option takes precedence over any destination specified in the *LPDEST* or *PRINTER* environment variables.

–n*copies*

Specifies the number of copies of the files to be written.

Lp expects that the files are appropriate for the printer — a printer expecting PostScript receives PostScript files, a printer expecting text files receives text. Use the **–b** option for raw binary files (such as sending PCL files to an HP-compatible printer).

DIAGNOSTICS

The *lp* utility exits 0 on success, and >0 if an error occurs.

ENVIRONMENT VARIABLES

Lp makes use of the following environment variables:

LPDEST

A default printer queue (in UNC format) which takes precedence over the one specified in *PRINTER*.

LP_PRINT

The location of the Windows NT utility **print.exe**. If this variable is set, *lp* uses this value to run **print.exe**; if it is not set, *lp* searches the current *PATH* and then checks **$WINDIR/system32** for the **print.exe** file.

PRINTER

The default printer queue (in UNC format). This queue is used unless *LPDEST* is set or the **–d** option is used.

NOTES

The *lp* command is a wrapper for the Windows NT Win32 **print.exe** command.

NAME

ls – list directory contents

SYNOPSIS

`ls [-AaCcDdFfgiLlnoqRrstu1] [`*`file ...`*`]`

DESCRIPTION

For each operand that names a *file* of a type other than directory, *ls* displays its name as well as any requested, associated information. For each operand that names a *file* of type directory, *ls* displays the names of files contained within that directory, as well as any requested, associated information.

If no operands are given, the contents of the current directory are displayed. If more than one operand is given, non-directory operands are displayed first; directory and non-directory operands are sorted separately and in lexicographical order.

The following options are available:

–A List all entries except for '.' and '..' .

–a Include directory entries whose names begin with a dot (.).

–C Force multi-column output; this is the default when output is to a terminal.

–c Use time when file status was last changed for sorting or printing.

–D Always display domain name in long formats. The default is to display the domain name only if it's different from the current domain.

–d Directories are listed as plain files (not searched recursively).

–F Display a slash (/) immediately after each pathname that is a directory, an asterisk (*) after each that is executable, and a vertical bar (|) after each that is a FIFO.

–f Output is not sorted.

–g List in long format (as for **–l**), but do not print the owner.

–i For each file, print the file's serial number.

–L If the *file* is a symbolic link, list the file referenced by the link rather than the link itself. Since INTERIX does not yet support symbolic links, this option has no effect.

–l (The lowercase letter ''ell.'') List in long format. (See below.) If the output is to a terminal, a total sum for all the file sizes is output on a line before the long listing.

–n List in long format (as for **–l**), but print the user ID and group ID numbers instead of the user and group names.

–o List in long format (as for **–l**), but do not print the group.

–q Force printing of non-graphic characters in file names as the character '?'; this is the default when output is to a terminal.

–R Recursively list subdirectories encountered.

–r Reverse the order of the sort to get reverse lexicographical order or the oldest entries first.

–s Display the number of file system blocks actually used by each file, in units of 512 bytes, where partial units are rounded up to the next integer value. If the output is to a terminal, a total sum for all the file sizes is output on a line before the listing.

–t Sort by time modified (most recently modified first) before sorting the operands by lexicographical order.

–u Use time of last access, instead of last modification of the file for sorting (**-t**) or printing (**-l**).

–1 (The numeric digit ''one.'') Force output to be one entry per line. This is the default when output is not to a terminal.

The **–1**, **–C**, and **–l** options all override each other; the last one specified determines the format used.

The **–c**, and **–u** options override each other; the last one specified determines the file time used.

By default, *ls* lists one entry per line to standard output; the exceptions are to terminals or when the **–C** option is specified.

File information is displayed with one or more <blank>s separating the information associated with the **–i**, **–s**, and **–l** options.

THE LONG FORMAT

If the **–l** option is given, the following information is displayed for each file: file mode, number of links, owner name, group name, number of bytes in the file, abbreviated month, day-of-month file was last modified, hour file last modified, minute file last modified, and the pathname. In addition, for each directory whose contents are displayed, the total number of 512-byte blocks used by the files in the directory is displayed on a line by itself immediately before the information for the files in the directory.

If the owner or group names are not a known user or group name the numeric ID's are displayed.

If the file is a character special or block special file, the major and minor device numbers for the file are displayed in the size field.

The file mode printed under the -l option consists of the entry type, owner permissions, and group permissions. The entry type character describes the type of file, as follows:

b Block special file.
c Character special file.
d Directory.
p FIFO or a Windows NT system file.
– Regular file.

The next three fields are three characters each: owner permissions, group permissions, and other permissions. Each field has three character positions:

1. If **r**, the file is readable; if **–**, it is not readable.
2. If **w**, the file is writeable; if **–**, it is not writeable.
3. The first of the following that applies:

S	If in the owner permissions, the file is not executable and set-user-ID mode is set. If in the group permissions, the file is not executable and set-group-ID mode is set.
s	If in the owner permissions, the file is executable and set-user-ID mode is set. If in the group permissions, the file is executable and setgroup-ID mode is set.
x	The file is executable or the directory is searchable.
–	The file is neither readable, writeable, executable, nor set-user-ID nor set-group-ID mode.

DIAGNOSTICS

The *ls* utility exits 0 on success, and >0 if an error occurs.

ENVIRONMENT VARIABLES

The following environment variables affect the execution of *ls*:

BLOCKSIZE
: If the environmental variable *BLOCKSIZE* is set, the block counts (see **–s**) will be displayed in units of that size block.

COLUMNS
: If this variable contains a string representing a decimal integer, it is used as the column position width for displaying multiple-text-column output. The *ls* utility calculates how many pathname text columns to display based on the width provided.

TZ
: The timezone to use when displaying dates.

SEE ALSO

chgrp, *chmod*, *chown*, *ln*, *touch*

NAME

mail – send or read mail

SYNOPSIS

`mail [-e] [-f file]`

`mail name ...`

DESCRIPTION

The *mail* utility lets you read or send mail messages. It is a minimal (possibly even skimpy) environment.

To read mail, invoke *mail* without any arguments. The *mail* utility opens your mailfile (or the mailfile specified with the **–f** option). The default mailfile is **$OPENNT_ROOT/usr/spool/mail/***name*, where *name* is your login name, including your domain name. Additional commands available during mail reading sessions are described below.

To send mail, invoke *mail* with the name of a user as the operand. The *mail* utility then takes standard input up until the end-of-file or a line consisting of only a . character and adds it to the mailfile of the user *name*.

The *mail* command takes the following options:

–e Test for new mail but display nothing. Exits with success if there is mail present, failure if there isn't.

–f *file* Read mail from the specified *file* instead of from the default mailbox.

On startup, the *mail* utility reads the startup file **$OPENNT_ROOT/etc/mail.rc**, which contains commands (as documented in the *mailx* reference page). It then reads the startup file **$HOME/.mailrc**.

Mail Reading Commands

The *mail* utility accepts the following commands:

<newline>
: Go on to next message.

<EOF> Store undeleted mail and quit.

!*command*
: Execute the named *command* in the shell.

+ Go on to next message (same as newline).

– Go back to previous message.

d Delete the message and go on to the next message.

m[*name*...]
: Mail the message to the user(s) named *name*; if no user specified, the message is mailed to the user who invoked *mail*.

p Display message again.

q Store undeleted mail and quit (same as EOF).

s[*file*] Save the message in *file*. If no *file* is named, save it in the file **~/mbox**.

w[*file*] Save the message without its headers in *file*. If no *file* is named, save it in the file **~/mbox**.

x Exit *mail*, undoing all changes to the mailfile in this session. Put all mail back in the mailfile and stop.

DIAGNOSTICS

The *mail* command can exit with one of the following statuses:

0 The user had mail and the session completed successfully.
1 The user had no mail or there was an initialization error.
>1 An error occurred after initialization.

ENVIRONMENT VARIABLES

The *mail* command makes use of the following environment variables, if set:

HOME The user's home directory.
TZ The current time zone; used in determining the setting of certain date and time strings.

FILES

The *mail* utility makes use of the following files:

$OPENNT_ROOT/usr/spool/mail/*login*
Default mailbox file.
$HOME/mbox
Default storage location for read mail messages.

NOTES

The *mail* and the *mailx* commands are the same image. If invoked as *mail*, it behaves as *mail*; if invoked as *mailx*, it behaves as *mailx*. When invoked as *mail*, it ignores the contents of the *mailx* startup files (**$OPENNT_ROOT/etc/mailx.rc** and **$HOME/.mailrc**).

The *mail* command by itself has no ability to send mail to users on other hosts on the network. That requires a mail routing agent and a mail transfer agent, neither of which are provided with INTERIX.

NOTES

The *mail* command is actually the *mailx* command; if invoked as *mail*, it behaves as described on this page.

In all other ways, *mail* behaves as though it were *mailx* with the following options set:

oneintr Abort the current letter on one interrupt rather than two.
replacedeadletter
Replace the **dead.letter** files instead of appending to them.
savelocal
Try to save the **dead.letter** file in the current directory rather than in the home directory.
set dot End message entry on a . character instead of ~.
set escape=
Disable tilde escapes.
set noheader
Don't print a message summary on startup.
set quiet Don't print other messages on startup.

See the *mailx* reference page for more details.

SEE ALSO

mailx

NAME

mailx – send and receive mail

SYNOPSIS

```
mailx [-iInv] [-s subject] [-c cc-addr] [-b bcc-addr] to-addr...
mailx -f [-HiInNv] [file]
mailx [-HiInNv] [-u user]
mailx -e
```

INTRODUCTION

Mailx is an electronic mail processing utility which is reminiscent of an line editor for mail, except that lines are replaced by messages. *Mailx* has two basic modes of operation: *reading* mail and *sending* mail; each mode has its own commands and options. The first synopsis line above is for sending mail.

To send mail, give the command *mailx* and one or more login names (for users on the same machine) or e-mail addresses; *mailx* prompts for a message (unless one has been supplied with the **–s** option) and then enters input mode. All text typed into *mailx* form part of the message until you enter the end-of-file character or the command **~.**. See "Sending and Replying to Mail" for a longer discussion; see "Tilde / Escapes" for a list of the commands used in input mode.

To read mail, give the command *mailx*; *mailx* displays a list of message headers, each representing a message in your system mailbox. In this mode, you can read, save and delete mail messages. You can also reply or send mail (dropping into input mode to do so), and set options to control *mailx*'s behavior.

Mailx takes these options.

–b*list* Send blind carbon copies to *list*. List should be a comma-separated list of names. Normally, all recipients can see who was sent a copy of the message; that information is suppressed in a blind carbon copy so that only the sender and the blind carbon copy recipient know about the copy.

–c*list* Send carbon copies to *list* of users.

–e Check for mail in the system mailbox and exit without printing a message. If there is mail in the system mailbox, *mailx* returns an exit status of 0; otherwise, it returns an exit status of 1. In a shell script or a startup file such as **.profile**, this is useful for checking for the presence of mail.

–f[*mbox***]** Read in the contents of your **mbox** (or the specified file) for processing; when you **quit**, *mailx* writes undeleted messages back to this file.

–H Write a header summary only; this is only the first set of 18 messages, starting with the first unread message.

–i Ignore tty interrupt signals. This is particularly useful when using *mailx* on noisy phone lines. You can also set this behavior inside *mailx* with the *ignore* option.

–I Run *mailx* in interactive mode even when input isn't a terminal. In particular, the ~ special character when sending mail is only active in interactive mode.

–n Inhibits reading the system startup files **$OPENNT_ROOT/etc/mail.rc** and **$OPENNT_ROOT/etc/mailx.rc** upon startup.

–N Inhibit the initial display of message headers when reading mail or editing a mail folder.

–s*subject* Specify subject on command line (only the first argument after the **–s** flag is used as a subject; be careful to quote subjects containing spaces.)

–u*user* Is equivalent to:

```
mailx -f $OPENNT_ROOT/usr/spool/mail/user
```

You must have the appropriate permissions to read mail for *user*.

–v Verbose mode. The details of delivery are displayed on the user's terminal.

Sending mail

To send a message to one or more people, *mailx* can be invoked with arguments which are the names of people to whom the mail will be sent. You are then expected to type in your message, followed by a control–D at the beginning of a line. The section "Replying to or originating mail" describes some features of *mailx* available to help you compose your letter.

An e-mail address can be either a local address or an Internet-style address. Internet-style addresses are of the form *name@domain*; all other addresses are taken as local addresses. (This means that *mailx* does not recognize UUCP-style bang addresses or the %-hack.)

Reading mail

To read mail, give *mailx* no arguments. It checks your mail out of the post office, then prints out a one-line header for each message found. Each message is numbered, and the message your commands affect is called the *current message*. When you start *mailx*, the current message is the first message, message 1. You can display the message using the **print** command (which can be abbreviated **p**). To make a new message the current message, you can use the commands **+** and **–** to move backwards and forwards in the list, or you can specify a message by number. There are other ways of specifying which messages your commands will act on; see "Specifying messages."

Disposing of mail

After you examine a message, you can **delete** (**d**) the message or **reply** (**r**) to it. Deletion causes the *mailx* program to forget about the message. This is not irreversible; the message can be **undeleted** (**u**) by giving its number, or the *mailx* session can be aborted by giving the **exit** (**x**) command (rather than the usual **quit** command). Usually, deleted messages disappear, never to be seen again.

Specifying messages

Commands such as **print** and **delete** can be given a list of message numbers as arguments to apply to a number of messages at once. Thus **delete 1 2** deletes messages 1 and 2, while **delete 1–5** deletes messages 1 through 5. The special name ***** addresses all messages, and **$** addresses the last message; thus the command **top** which prints the first few lines of a message could be used in "top *" to print the first few lines of all messages.

Message specifications are summarized here:

num	Message number *num*
n–m	Messages number *n* through *m*
*****	All messages
.	The current message
+	The following appropriate message
–	The previous appropriate message
^	The first appropriate message in the list
$	The last message in the list
address	All messages from *address* (as shown in the header)
/string	All messages with *string* in the subject line (case is ignored)
:*c*	All messages of type *c*, where *c* is one of: **d** Deleted messages **n** New messages **o** Old messages (messages that aren't read or new) **r** Read messages **u** Unread messages

The word "appropriate" is needed because while most commands work only on undeleted commands, the **undelete** command only works on deleted commands. So **print ^** prints the first undeleted message in the list, while **undelete ^** undeletes the first deleted message in the list.

Sending new mail or replying to mail

You can use the **reply** command to set up a response to a message, sending it back to the person who it was from. Text you then type in, up to an end-of-file, defines the contents of the message. While you are composing a message, *mailx* treats lines beginning with the character **~** specially. For instance, typing **~m** (alone on a line) will place a copy of the current message into the response right shifting it by a tabstop (see *indentprefix* variable, below). Other escapes will set up subject fields, add and delete recipients to the message and allow you to escape to an editor to revise the message or to a shell to run some

commands. (These options are given in the summary below.)

Ending a mail processing session

You can end a *mailx* session with the **quit** (**q**) command. Messages which have been examined go to your *mbox* file unless they have been deleted in which case they are discarded. Unexamined messages go back to the post office. (See the **–f** option above).

Personal and systemwide distribution lists

It is also possible to create a personal distribution lists so that, for instance, you can send mail to "cohorts" and have it go to a group of people. Such lists can be defined by placing a line like

```
alias cohorts bill ozalp jkf mark kridle@ucbcory
```

in the file **.mailrc** in your home directory. The current list of such aliases can be displayed with the **alias** command in *mailx*. In mail you send, personal aliases will be expanded in mail sent to others so that they will be able to **reply** to the recipients.

Mailx has a number of options which can be set in the **.mailrc** file to alter its behavior; thus "set askcc" enables the *askcc* feature. (These options are summarized below.)

COMMAND SUMMARY

Each command is typed on a line by itself, and may take arguments following the command word. The general syntax is:

command **[** *msg-list* **] [** *arguments* **]**

The command need not be typed in its entirety — the first command which matches the typed prefix is used. For commands which take message lists as arguments, if no message list is given, then *mailx* uses the next message forward which satisfies the command's requirements. If there are no messages forward of the current message, the search proceeds backwards, and if there are no good messages at all, *mailx* types "No applicable messages" and aborts the command.

– Print out the preceding message. If given a numeric argument *n*, goes to the *n*'th previous message and prints it.

? Print a brief summary of commands.

! Execute the shell (see *sh* and *csh*) command which follows.

Print (**P**) Like **print** but also print out ignored header fields. (See also **print**, **ignore** and **retain**.)

Reply (**R**) Reply to sender. Does not reply to other recipients of the original message.

Type (**T**) Identical to the **Print** command.

alias (**a**) With no arguments, print out all currently-defined aliases. With one argument, print out that alias. With more than one argument, create a new alias or change an old one.

alternates
(**alt**) The **alternates** command is useful if you have accounts on several machines. It can be used to inform *mailx* that the listed

addresses are really you. When you **reply** to messages, *mailx* will not send a copy of the message to any of the addresses listed on the **alternates** list. If the **alternates** command is given with no argument, the current set of alternate names is displayed.

chdir (**c**) Changes the user's working directory to that specified, if given. If no directory is given, then changes to the directory specified in *HOME*.

copy (**co**) Like **save**, the **copy** command saves a file, except that it does not mark the saved messages for deletion when you quit.

delete (**d**) Mark the messages in the list as deleted. Deleted messages will not be saved in *mbox*, nor will they be available for most other commands. They are not actually deleted until you **quit** *mailx*. See **undelete**.

dp (also **dt)** Delete the current message and prints the next message. If there is no next message, *mailx* says "at EOF".

edit (**e**) Edit each message in the supplied message list, using the editor specified in the *EDITOR* environment variable. On return from the editor, the message is read back in.

exit (**ex** or **x**) Effects an immediate return to the shell without modifying the user's system mailbox, his *mbox* file, or his edit file in **–f**.

file (**fi**) The same as **folder**.

folders List the names of the folders in your folder directory.

folder (**fo**) The **folder** command switches to a new mail file or folder. With no arguments, it tells you which file you are currently reading. If you give it an argument, it will write out changes (such as deletions) you have made in the current file and read in the new file. Some special conventions are recognized for the name.

#	the previous file
%	your system mailbox
%*user*	*user*'s system mailbox
&	your mbox file
+*file*	the named *file* in your folder directory (see the *folder* option)

from (**f**) Print the message headers of the messages in a list.

headers

(**h**) List the current range of headers, which is an 18–message group. The argument **+** displays the next 18-message group, and the argument **–** displays the previous 18-message group.

help A synonym for **?**

hold (**ho**, also **preserve**) Marks each message in a supplied list to be saved in your system mailbox instead of in *mbox*. Doesn't override the **delete** command.

ignore Add the supplied list of header fields to the *ignored* list. Header fields in the ignore list are not printed on your terminal when you print a message. This command is very handy for suppression of certain machine-generated header fields. The commands **Type** and **Print** display a message in its entirety, including ignored fields. If no arguments are given, **ignore** lists the current set of ignored fields.

mail (**m**) Send mail to the names specified as arguments; the names can be login names or distribution group names.

mbox Indicate that a list of messages be sent to **mbox** in your home directory when you quit. This is the default action for messages if you do *not* have the **hold** option set.

next (**n**, like **+** or pressing Enter). Goes to the next message in sequence and types it. With an argument list, types the next matching message.

preserve
(**pre**) A synonym for **hold**.

print (**p**) Display on the user's terminal each message in the message list.

quit (**q**) Terminate the session, saving all undeleted, unsaved messages in the user's *mbox* file in his login directory, preserving all messages marked with **hold** or **preserve** or never referenced in his system mailbox, and removing all other messages from his system mailbox. If new mail has arrived during the session, the message "You have new mail" is given. If you **quit** while editing a mailbox file with the **–f** flag, then *mailx* rewrites file you're editing, and returns you to the shell, unless the file rewrite fails. If it fails, you can escape *mailx* with the **exit** command.

reply (**r**) Reply to a message (or message list); the reply is sent to the sender and all recipients of the specified message(s). The default message must not be deleted.

respond
A synonym for **reply**.

retain Add the list of header fields named to the *retained*list Only the header fields in the retain list are shown on your terminal when you print a message. All other header fields are suppressed. The **Type** and **Print** commands can be used to print a message in its entirety. If **retain** is executed with no arguments, it lists the current set of retained fields.

save (**s**) Append each message in a message list to the end of a specified file. *Mailx* echoes to the terminal the filename, the line count, and the character count.

set (**se**) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form *option=value* (no space before or after =) or *option*. Quotation marks may be placed around any part of the assignment statement to quote blanks or tabs, i.e.

```
set indentprefix="->"
```

saveignore
Saveignore is to **save** what **ignore** is to **print** and **type**. Header fields thus marked are filtered out when saving a message by **save** or when automatically saving to *mbox*.

saveretain
Saveretain is to **save** what **retain** is to **print** and **type**. Header fields thus marked are the only ones saved with a message when saving by **save** or when automatically saving to *mbox*. **Saveretain** overrides **saveignore**.

shell (**sh**) Invokes an interactive version of the shell.

size Takes a message list and prints out the size in characters of each message.

source The **source** command reads commands from a file.

top Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable **toplines** and defaults to

five.

type (**t**) A synonym for **print**.

unalias Takes a list of names defined by **alias** commands and discards the remembered groups of users. The group names no longer have any significance.

undelete
(**u**) Takes a message list and marks each message as **not** being deleted.

unread (**U**) Takes a message list and marks each message as **not** having been read.

unset Takes a list of option names and discards their remembered values; the inverse of **set**.

visual (**v**) Takes a message list and invokes the display editor on each message.

write (**w**) Similar to **save**, except that **only** the message body (*without* the header) is saved. Extremely useful for such tasks as sending and receiving source program text over the message system.

xit (**x**) A synonym for **exit**.

z *Mailx* presents message headers in windowfuls as described under the **headers** command. You can move *mailx*'s attention forward to the next window with the **z** command. Also, you can move to the previous window by using **z–**.

Tilde/Escapes

Here is a summary of the tilde escapes, which are used when composing messages to perform special functions. Tilde escapes are only recognized at the beginning of lines. The name "tilde escape" is somewhat of a misnomer since the actual escape character can be set by the option **escape**.

~! *command*
Execute the indicated shell command, then return to the message.

~b *name ...*
Add the given names to the list of carbon copy recipients but do not make the names visible in the Cc: line ("blind" carbon copy).

~c *name ...*
Add the given names to the list of carbon copy recipients.

~d Read the file **dead.letter** from your home directory into the message.

~e Invoke the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.

~f *messages*
Read the named messages into the message being sent. If no messages are specified, read in the current message. Message headers currently being ignored (by the **ignore** or **retain** command) are not included.

~F *messages*
Identical to **~f**, except all message headers are included.

~h Edit the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field by using the current terminal erase and kill characters.

~m *messages*
Read the named messages into the message being sent, indented by a

tab or by the value of *indentprefix*. If no messages are specified, read the current message. Message headers currently being ignored (by the **ignore** or **retain** command) are not included.

~M *messages*
Identical to **~m**, except all message headers are included.

~p Print out the message collected so far, prefaced by the message header fields.

~q Abort the message being sent, copying the message to **dead.letter** in your home directory if **save** is set.

~r *filename*
Read the named file into the message.

~s *string* Cause the named string to become the current subject field.

~t *name ...*
Add the given names to the direct recipient list.

~v Invoke an alternate editor (defined by the *VISUAL* option) on the message collected so far. Usually, the alternate editor will be a screen editor. After you quit the editor, you may resume appending text to the end of your message.

~w *filename*
Write the message onto the named file.

~| *command*
Pipe the message through the command as a filter. If the command gives no output or terminates abnormally, retain the original text of the message. The command *fmt* is often used as **command** to rejustify the message.

~: *mail-command*
Execute the given mail command. Not all commands, however, are allowed.

~~ *string* Insert the string of text in the message prefaced by a single ~. If you have changed the escape character, then you should double that character in order to send it.

Mail Options

Options are controlled via **set** and **unset** commands. Options may be either binary, in which case it is only significant to see whether they are set or not; or string, in which case the actual value is of interest. The binary options include the following:

append Causes messages saved in *mbox* to be appended to the end rather than prepended. This should always be set (perhaps in **$OPENNT_ROOT/etc/mail.rc** or **$OPENNT_ROOT/etc/mailx.rc**).

ask Causes *mailx* to prompt you for the subject of each message you send (unless a message has been specified with **–s**). If you respond with simply a newline, no subject field will be sent. This is synonymous with **asksub**.

askbcc Causes *mailx* to prompt you for the blind copy list.

askcc Causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.

asksub Causes *mailx* to prompt you for the subject of each message you send (unless a message has been specified with **–s**). If you respond with

simply a newline, no subject field will be sent. This is synonymous with **ask**.

autoprint
Causes the **delete** command to behave like **dp** — thus, after deleting a message, the next one will be typed automatically.

bang Causes *mailx* to treat specially exclamation marks in escape command lines. (See the **escape** command.) The default is off.

cmd=*command*
Sets a *command* as the default command invoked by the **pipe** command.

debug Setting the binary option *debug* is the same as specifying **–d** on the command line and causes *mailx* to output all sorts of information useful for debugging *mailx*.

dot The binary option *dot* causes *mailx* to interpret a period alone on a line as the terminator of a message you are sending.

flipr Reverses the meaning of the **R** and **r** commands.

header Causes *mailx* to display a header summary when entered in read mode. The default is **header**.

hold This option is used to hold messages in the system mailbox by default.

ignore Causes interrupt signals from your terminal to be ignored and echoed as @'s.

ignoreeof
An option related to *dot* is *ignoreeof* which makes *mailx* refuse to accept a control-d as the end of a message. *Ignoreeof* also applies to *mailx* command mode.

keep Causes *mailx* to truncate an empty mailbox, system mailbox or **mbox** to zero length instead of removing it.

keepsave
Keep messages in the system mailbox even if they have already been saved in other files. The default is **nokeepsave**.

metoo Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.

noheader
Setting the option *noheader* is the same as giving the **–N** flag on the command line.

nosave Normally, when you abort a message with two RUBOUT characters (erase or delete) *mailx* copies the partial letter to the file **dead.letter** in your home directory. Setting the binary option *nosave* prevents this.

oneintr Abort letter on one interrupt instead of two. See also the **ignore** option and the **–i** option.

outfolder
Causes *mailx* to store the files of outgoing messages in the relative directory specified by the **folder** variable. This has no effect if the value of **folder** is an absolute pathname.

page Inserts a form-feed after each message sent through the pipe created by the **pipe** command.

prompt=*string*
Sets the command-mode prompt to *string*. The default is "? ". If the string is null, or **noprompt** is set, *mailx* does not prompt.

quiet Suppresses the printing of the version when first invoked.

record=*file*
Records all outgoing mail in *file*; the default is **norecord**. If **outfolder** is set, *file* is treated as a directory which will contain files with outgoing mail.

replacedeadletter
If set, the existing dead-letter file is replaced by the cancelled mail rather than appending the cancelled mail to the existing dead-letter file.

Replyall
Reverses the sense of **reply** and **Reply** commands.

save Forces *mailx* to save messages in the dead-letter file (specified by *DEAD*) if they are interrupted or there are delivery errors.

savelocal
If set, *mailx* tries to save the dead-letter file in the current directory instead of in the home directory.

screen=*number*
Sets the number of lines in a screenful of headers. This value is used by the **headers** and **z** commands in mail-reading mode. If not set, *mailx* determines the current window size.

searchheaders
If this option is set, then a message-list specifier in the form ''/x:y'' will expand to all messages containing the substring ''y'' in the header field ''x''. The string search is case insensitive.

sendmail=*command*
Specifies that *command* be used as the command for delivering mail messages. For local delivery (useful for automatic mail generated by utilities such as *at*), set this value to "ssimda".

show-rcpt
Specifies that the recipient, not the sender, is displayed in header summaries.

showto When displaying a header summary, displays the recipient instead of the sender for any messages which you sent.

Sign Contains a string that is inserted into the message when you give the **~A** command.

sign Contains a string that is inserted into the message when you give the **~a** command.

verbose
Setting the option *verbose* is the same as using the **–v** flag on the command line. When mail runs in verbose mode, the actual delivery of messages is displayed on the user's terminal.

Option String Values

EDITOR
Pathname of the text editor to use in the **edit** command and **~e** escape. If not defined, then a default editor is used.

LISTER Pathname of the directory lister to use in the **folders** command. Default is **$OPENNT_ROOT/bin/ls**.

PAGER Pathname of the program to use in the **more** command or when *crt* variable is set. If this option is not defined, *more* is used.

SHELL Pathname of the shell to use in the **!** command and the **~!** escape. A default shell is used if this option is not defined.

VISUAL
Pathname of the text editor to use in the **visual** command and **~v** escape.

crt The valued option *crt* is used as a threshold to determine how long a message must be before *PAGER* is used to read it. If *crt* is set without a value, then the height of the terminal screen stored in the system is used to compute the threshold (see *stty*).

escape=c If defined, the character *c* is used in the place of ~ to denote escapes.

folder The name of the directory to use for storing folders of messages. If this name begins with a slash (/), *mailx* considers it to be an absolute pathname; otherwise, the folder directory is found relative to your home directory.

MBOX The name of the *mbox* file. It can be the name of a folder. The default is **mbox** in the user's home directory.

record If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not so saved.

indentprefix
String used by the **~m** tilde escape for indenting messages, in place of the default tab character (^I). Be sure to quote the value if it contains spaces or tabs.

toplines The number of lines to be printed with the **top** command; normally, the first five lines are printed.

ENVIRONMENT VARIABLES

Mailx uses the following environment variables:

DEAD The name of the file in which to save interrupted or cancelled mail. The default is **$HOME/dead.letter**.

EDITOR
The editor to use when the **edit** or **~e** commands are given. Default is *vi*.

HOME The user's home directory.

LISTER The command used for listing the contents of the *folder* directory when the **folders** command is given. Defaults to *ls*.

MAILRC
The personal startup file to use. Defaults to **$HOME/.mailrc**.

MBOX The file used to save read messages. Defaults to **$HOME/mbox**.

PAGER The command used to display a mail message if the number of lines in the message is greater than the value of the *crt* variable.

SHELL The command interpreter to use for the **!** command. Defaults to *sh*.

TERM The name of the terminal type. If the variable *screen* hasn't been set, *mailx* uses the description of the terminal to determine the number of lines on the screen.

VISUAL
The name of the editor to use when the **visual** or **~v** commands are given. Defaults to *vi*.

USER Login name of user.

FILES

$OPENNT_ROOT/usr/spool/mail/*
The post office; new mail is delivered to this directory.

~/mbox User's old mail.

$HOME/.mailrc
Personal startup file. Commands in this file are executed after commands in **$OPENNT_ROOT/etc/mail.rc** and **$OPENNT_ROOT/etc/mailx.rc**.

$OPENNT_ROOT/tmp/R*
Temporary files.

$OPENNT_ROOT/usr/share/misc/Mail.*help
Help files.

$OPENNT_ROOT/etc/mail.rc
System initialization file. This file is also read if *mailx* is invoked as *mail*.

$OPENNT_ROOT/etc/mailx.rc
System initialization file. This file is only read if *mailx* is invoked as *mailx*. Commands in this file supersede those in **$OPENNT_ROOT/etc/mail.rc**, but may be superseded by those in the user's personal startup file, **$HOME/.mailrc**.

SEE ALSO

fmt, *mail*, *ssimda*

The Mail Reference Manual.

NAME

make – maintain program dependencies

SYNOPSIS

```
make [-eiknqrSst ] [-D variable] [-d flags]
     [-f makefile] [-I directory] [-m directory]
     [-V variable] [variable=value] [target ... ]
```

DESCRIPTION

Make is a program designed to simplify the maintenance of other programs. Its input is a list of specifications as to the files upon which programs and other files depend. If the file **makefile** exists, it is read for this list of specifications. If it does not exist, the file **Makefile** is read. If the file **.depend** exists, it is read.

The options are as follows:

–D *variable*
Define *variable* to be 1, in the global context.

–d *flags* Turn on debugging, and specify which portions of *make* are to print debugging information. *Flags* is one or more of the following:

- **A** Print all possible debugging information; equivalent to specifying all of the debugging flags.
- **a** Print debugging information about archive searching and caching.
- **c** Print debugging information about conditional evaluation.
- **d** Print debugging information about directory searching and caching.
- **g1** Print the input graph before making anything.
- **g2** Print the input graph after making everything, or before exiting on error.
- **j** Print debugging information about running multiple shells.
- **m** Print debugging information about making targets, including modification dates.
- **s** Print debugging information about suffix-transformation rules.
- **t** Print debugging information about target list maintenance.
- **v** Print debugging information about variable assignment.

–e Specify that environmental variables override macro assignments within makefiles.

–f *makefile*
Specify a makefile to read instead of the default **makefile** and **Makefile**. If *makefile* is **–** standard input is read. Multiple makefiles may be specified, and are read in the order specified.

–I *directory*
Specify a directory in which to search for makefiles and included makefiles. The system makefile directory is automatically included as part of this list.

–i Ignore non-zero exit of shell commands in the makefile. Equivalent to specifying **–** before each command line in the makefile.

–k Continue processing after errors are encountered, but only on those targets that do not depend on the target whose creation caused the error.

–m *directory*
Specify a directory to search for **sys.mk** and makefiles included via the <...> style **.include** directive. Multiple directories can be added to form a search path. This path will override the default system include path (**$OPENNT_ROOT/usr/share/mk**). Furthermore the system include path will be appended to the search path used for "..."-style inclusions (see the **–I** option).

–n Display the commands that would have been executed, but do not actually execute them.

–q Do not execute any commands, but exit 0 if the specified targets are up-to-date and 1, otherwise.

–r Do not use the built-in rules specified in the system makefile.

–S A synonym for **–k**.

–s Do not echo any commands as they are executed. Equivalent to specifying @ before each command line in the makefile.

–t Rather than re-building a target as specified in the makefile, create it or update its modification time to make it appear up-to-date.

–V *variable*
Print *make*'s idea of the value of *variable*, in the global context. Do not build any targets. Multiple instances of this option may be specified. The variables will be printed one per line with a blank line for each null or undefined variable.

variable=value
Set the value of the variable *variable* to *value*.

There are seven different types of lines in a makefile: file dependency specifications, shell commands, variable assignments, include statements, conditional directives, for loops, and comments.

In general, lines may be continued from one line to the next by ending them with a backslash (\). The trailing newline character and initial whitespace on the following line are compressed into a single space.

FILE DEPENDENCY SPECIFICATIONS

Dependency lines consist of one or more targets, an operator, and zero or more sources. This creates a relationship where the targets "depend" on the sources and are usually created from them. The exact relationship between the target and the source is determined by the operator that separates them. The three operators are as follows:

: A target is considered out-of-date if its modification time is less than those of any of its sources. Sources for a target accumulate over dependency lines when this operator is used. The target is removed if

make is interrupted.

! Targets are always re-created, but not until all sources have been examined and re-created as necessary. Sources for a target accumulate over dependency lines when this operator is used. The target is removed if *make* is interrupted.

:: If no sources are specified, the target is always re-created. Otherwise, a target is considered out-of-date if any of its sources has been modified more recently than the target. Sources for a target do not accumulate over dependency lines when this operator is used. The target will not be removed if *make* is interrupted.

Targets and sources may contain the shell wildcard values ?, *, () and {}. The values ?, * and [] may only be used as part of the final component of the target or source, and must be used to describe existing files. The value {} need not necessarily be used to describe existing files. Expansion is in directory order, not alphabetically as done in the shell.

SHELL COMMANDS

Each target may have associated with it a series of shell commands, normally used to create the target. Each of the commands in this script *must* be preceded by a tab. While any target may appear on a dependency line, only one of these dependencies may be followed by a creation script, unless the **::** operator is used.

If the first or first two characters of the command line are **@** and / or **–**, the command is treated specially. A **@** causes the command not to be echoed before it is executed. A **–** causes any non-zero exit status of the command line to be ignored.

VARIABLE ASSIGNMENTS

Variables in make are much like variables in the shell, and, by tradition, consist of all upper-case letters. The five operators that can be used to assign values to variables are as follows:

= Assign the value to the variable. Any previous value is overridden.

+= Append the value to the current value of the variable.

?= Assign the value to the variable if it is not already defined.

:= Assign with expansion, i.e. expand the value before assigning it to the variable. Normally, expansion is not done until the variable is referenced.

!= Expand the value and pass it to the shell for execution and assign the result to the variable. Any newlines in the result are replaced with spaces.

Any white-space before the assigned *value* is removed; if the value is being appended, a single space is inserted between the previous contents of the variable and the appended value.

Variables are expanded by surrounding the variable name with either curly braces (**{}**) or parenthesis (**()**) and preceding it with a dollar sign (**$**.) If the variable name contains only a single letter, the surrounding braces or parenthesis are not required. This shorter form is not recommended.

Variable substitution occurs at two distinct times, depending on where the variable is being used. Variables in dependency lines are expanded as the line

is read. Variables in shell commands are expanded when the shell command is executed.

The four different classes of variables (in order of increasing precedence) are:

1. **Environment variables** Variables defined as part of *make*'s environment.
2. **Global variables** Variables defined in the makefile or in included makefiles.
3. **Command line variables** Variables defined as part of the command line.
4. **Local variables** Variables that are defined specific to a certain target. There are seven local variables; they have the highest precedence. They are shown here with the **$** prefix. The seven local variables are as follows:

   $> The list of all sources for this target; also known as **.ALLSRC.**

   .ARCHIVE
   The name of the archive file.

   $< The name / path of the source from which the target is to be transformed (the "implied" source); also known as **.IMPSRC.** It is evaluated only for inference rules.

   $% The name of the archive member (also known as **.MEMBER).** This only applies when the current target is an archive library member of the form *libname*(*member***.o***)*.

   $? The list of sources for this target that were deemed newer than the target; also known as **.OODATE.**

   $* The file prefix of the file, containing only the file portion, no suffix or preceding directory components; also known as **.PREFIX.**

   $@ The name of the target; also known as **.TARGET.** When building a library, it evaluates to the name of the library; see **$%**.

   The short forms (**@**, **?**, **>** and *****) are the ones specified by POSIX; the use of the longer terms is a BSD extension.

   You can modify the **@**, **<** and ***** variables with the letter D or F: D refers to the directory portion of the target name, and F refers to the target portion. For example, for the target "/ / C / Interix / bin / construct", ${@D} is "/ / C / Interix / bin" and @F is "construct".

Four of the local variables may be used in sources on dependency lines because they expand to the proper value for each target on the line. These variables are **@** (or **.TARGET**), ***** (or **.PREFIX**), **%** (or **.MEMBER**), and **.ARCHIVE**.

In addition, *make* sets or knows about the following variables:

*$* A single dollar sign **$**, i.e. $$ expands to a single dollar sign.

.MAKE The name used to execute *make* (`argv[0]`)

.CURDIR
A path to the directory where *make* was executed.

.OBJDIR
A path to the directory where the targets are built. At startup, *make* searches for an alternate directory to place target files and attempts to change into this special directory. In order, *make* tries to change into the directory named by the environment variable *MAKEOBJDIR*; if that fails, it tries to change into the directory named **obj.$MACHINE**

(if the environment variable *MACHINE* is not set, *make* calls the *uname*( ) function to determine the type of the machine). If it still has found no special directory, it tries the directory named **obj**. Finally, if none of the above directories are available *make* uses the current directory.

MAKEFLAGS
: The environment variable *MAKEFLAGS* may contain anything that may be specified on *make*'s command line. Anything specified on *make*'s command line is appended to the *MAKEFLAGS* variable which is then entered into the environment for all programs which *make* executes.

PWD An alternate path to the current directory. *Make* normally sets *.CURDIR* to the canonical path given by *getcwd*( ). However, if the environment variable *PWD* is set and gives a path to the current directory, then *make* sets *.CURDIR* to the value of *PWD* instead. Setting this variable is most useful on systems which have automounters, where the same directory may have different paths on different machines. *PWD* is set to the value of *.OBJDIR* and exported for all programs which *make* executes.

Variable expansion may be modified to select or modify each word of the variable (where a "word" is white-space delimited sequence of characters). The general format of a variable expansion is as follows:

```
{variable[:modifier[:...]]}
```

Each modifier begins with a colon and one of the following special characters. The colon may be escaped with a backslash (\).

C/*old_pattern*/*new_pattern*/**[g]**
: Modify the first occurrence of *old_pattern* in each word to be replaced with *new_pattern*. The *old_pattern* is a regular expression and the *new_pattern* is an *ed*-style replacement string. Normally, the first occurrence of the pattern in each word of the value is changed. The **1** modifier causes substitution to apply to at most one word; the **g** modifier causes the substitution to apply to as many instances of the search pattern as occur in the word or words it is found in. The **1** and **g** modifiers are orthogonal; **1** specifies whether multiple words are potentially affected, while **g** specifies whether multiple substitutions can occur in an affected word. If *old_pattern* begins with a carat (^,) *old_pattern* is anchored at the beginning of each word. If *old_pattern* ends with a dollar sign ($), it is anchored at the end of each word. Any character may be used as a delimiter for the parts of the modifier string. The anchoring, ampersand and delimiter characters may be escaped with a backslash (\).

  Variable expansion occurs in the normal fashion inside both *old_string* and *new_string* with the single exception that a backslash is used to prevent the expansion of a dollar sign ($) not a preceding dollar sign as is usual.

E Replaces each word in the variable with its suffix.

H Replaces each word in the variable with everything but the last component.

M *pattern*

Select only those words that match the rest of the modifier. The standard shell wildcard characters *, ?, and [] may be used. The wildcard characters may be escaped with a backslash (\).

N *pattern*
This is identical to **–M**, but selects all words which do not match the rest of the modifier.

Q Quotes every shell meta-character in the variable, so that it can be passed safely through recursive invocations of *make*.

R Replaces each word in the variable with everything but its suffix.

S/*old_pattern***/***new_pattern***/[g]**
Modify the first occurrence of *old_pattern* in each word to be replaced with *new_pattern*. If a g is appended to the last slash of the pattern, all occurrences in each word are replaced. If *old_pattern* begins with a carat (^,) *old_pattern* is anchored at the beginning of each word. If *old_pattern* ends with a dollar sign ($), it is anchored at the end of each word. Inside *new_string*, an ampersand (&) is replaced by *old_pattern*. Any character may be used as a delimiter for the parts of the modifier string. The anchoring, ampersand and delimiter characters may be escaped with a backslash (\).

Variable expansion occurs in the normal fashion inside both *old_string* and *new_string* with the single exception that a backslash is used to prevent the expansion of a dollar sign ($) not a preceding dollar sign as is usual.

T Replaces each word in the variable with its last component.

*old_string***=***new_string*
This is the AT&T System V style variable substitution. It must be the last modifier specified. If *old_string* or *new_string* do not contain the pattern matching character *%* then it is assumed that they are anchored at the end of each word, so only suffixes or entire words may be replaced. Otherwise *%* is the substring of *old_string* to be replaced in *new_string*

INCLUDE STATEMENTS, CONDITIONALS AND FOR LOOPS

Makefile inclusion, conditional structures and for loops reminiscent of the C programming language are provided in *make*. All such structures are identified by a line beginning with a single dot (.) character. Files are included with either:

```
.include <file>
```

or

```
.include "file"
```

Variables between the angle brackets or double quotes are expanded to form the file name. If angle brackets are used, the included makefile is expected to be in the system makefile directory. If double quotes are used, the including makefile's directory and any directories specified using the **–I** option are searched before the system makefile directory.

Conditional expressions are also preceded by a single dot as the first character of a line. The possible conditionals are as follows:

.undef *variable*

Un-define the specified global variable. Only global variables may be un-defined.

.if [**!**] *expression* [*operator expression* ...]
Test the value of an expression.

.ifdef [**!**] *variable* [*operator variable* ...]
Test the value of a variable.

.ifndef [**!**] *variable* [*operator variable* ...]
Test the value of a variable.

.ifmake [**!**] *target* [*operator target* ...]
Test the the target being built.

.ifnmake [**!**] *target* [*operator target* ...]
Test the target being built.

.else Reverse the sense of the last conditional.

.elif [**!**] *expression* [*operator expression* ...]
A combination of **.else** followed by **.if**.

.elifdef [**!**]*variable* [*operator variable* ...]
A combination of **.else** followed by **.ifdef**.

.elifndef [**!**] *variable* [*operator variable* ...]
A combination of **.else** followed by **.ifndef**.

.elifmake [**!**]*target* [*operator target* ...]
A combination of .else followed by **.ifmake**.

.elifnmake [**!**]*target* [*operator target* ...]
A combination of **.else** followed by **.ifnmake**.

.endif End the body of the conditional.

The *operator* may be any one of the following:

|| logical OR

&& Logical AND; of higher precedence than "".

As in C, *make* will only evaluate a conditional as far as is necessary to determine its value. Parentheses may be used to change the order of evaluation. The boolean operator **!** may be used to logically negate an entire conditional. It is of higher precedence than **&&**.

The value of *expression* may be any of the following:

defined Takes a variable name as an argument and evaluates to true if the variable has been defined.

make Takes a target name as an argument and evaluates to true if the target was specified as part of *make*'s command line or was declared the default target (either implicitly or explicitly, see **.MAIN**) before the line containing the conditional.

empty Takes a variable, with possible modifiers, and evaluates to true if the expansion of the variable would result in an empty string.

exists Takes a file name as an argument and evaluates to true if the file exists. The file is searched for on the system search path (see **.PATH**).

target Takes a target name as an argument and evaluates to true if the target has been defined.

Expression may also be an arithmetic or string comparison. Variable expansion is performed on both sides of the comparison, after which the integral values are compared. A value is interpreted as hexadecimal if it is preceded by 0x, otherwise it is decimal; octal numbers are not supported. The standard C

relational operators are all supported. If after variable expansion, either the left or right hand side of a **==** or **!=** operator is not an integral value, then string comparison is performed between the expanded variables. If no relational operator is given, it is assumed that the expanded variable is being compared against 0.

When *make* is evaluating one of these conditional expression, and it encounters a word it doesn't recognize, either the **make** or **defined** expression is applied to it, depending on the form of the conditional. If the form is **.ifdef** or **.ifndef**, the "defined" expression is applied. Similarly, if the form is **.ifmake** or **.ifnmake**, the "make" expression is applied.

If the conditional evaluates to true the parsing of the makefile continues as before. If it evaluates to false, the following lines are skipped. In both cases this continues until a **.else** or **.endif** is found.

For loops are typically used to apply a set of rules to a list of files. The syntax of a for loop is:

```
.for variable in expression
<make-rules>
.endfor
```

After the **for** expression is evaluated, it is split into words. The iteration variable is successively set to each word, and substituted in the make-rules inside the body of the **for** loop.

COMMENTS

Comments begin with a hash (**#**) character, anywhere but in a shell command line, and continue to the end of the line.

SPECIAL SOURCES

.IGNORE
Ignore any errors from the commands associated with this target, exactly as if they all were preceded by a dash (**–**).

.MADE Mark all sources of this target as being up-to-date.

.MAKE Execute the commands associated with this target even if the **–n** or **–t** options were specified. Normally used to mark recursive *make*s.

.NOTMAIN
Normally *make* selects the first target it encounters as the default target to be built if no target was specified. This source prevents this target from being selected.

.OPTIONAL
If a target is marked with this attribute and *make* can't figure out how to create it, it will ignore this fact and assume the file isn't needed or already exists.

.PRECIOUS
When *make* is interrupted, it removes any partially made targets. This source prevents the target from being removed.

.SILENT
Do not echo any of the commands associated with this target, exactly as if they all were preceded by an at sign (**@**.)

.USE Turn the target into *make*'s. version of a macro. When the target is used as a source for another target, the other target acquires the

commands, sources, and attributes (except for **.USE**) of the source. If the target already has commands, the **.USE** target's commands are appended to them.

.WAIT If special **.WAIT** source apppears in a dependency line, the sources that precede it are made before the sources that succeed it in the line. Loops are not detected except that targets that form loops are silently ignored.

SPECIAL TARGETS

Special targets may not be included with other targets, i.e. they must be the only target specified.

.BEGIN

Any command lines attached to this target are executed before anything else is done.

.DEFAULT

This is sort of a **.USE** rule for any target (that was used only as a source) that *make* can't figure out any other way to create. Only the shell script is used. The **.IMPSRC** variable of a target that inherits the **.DEFAULT** commands is set to the target's own name.

.END Any command lines attached to this target are executed after everything else is done.

.IGNORE

Mark each of the sources with the **.IGNORE** attribute. If no sources are specified, this is the equivalent of specifying the **–i** option.

.INTERRUPT

If *make* is interrupted, the commands for this target will be executed.

.MAIN If no target is specified when *make* is invoked, this target will be built.

.MAKEFLAGS

This target provides a way to specify flags for *make* when the makefile is used. The flags are as if typed to the shell, though the **–f** option will have no effect.

.ORDER

The named targets are made in sequence.

.PATH The sources are directories which are to be searched for files not found in the current directory. If no sources are specified, any previously specified directories are deleted.

.PHONY

Named sources have the **.PHONY** attribute. These targets are always considered to be out of date.

.PRECIOUS

Apply the **.PRECIOUS** attribute to any specified sources. If no sources are specified, the **.PRECIOUS** attribute is applied to every target in the file.

.SILENT

Apply the **.SILENT** attribute to any specified sources. If no sources are specified, the **.SILENT** attribute is applied to every command in the file.

.SUFFIXES

Each source specifies a suffix to *make*. If no sources are specified, any previous specified suffices are deleted.

ENVIRONMENT VARIABLES

Make utilizes the following environment variables, if they exist:

MACHINE
: The name of the machine. Used by the *.OBJDIR* variable. If not specified, the machine type returned by *uname*( ) is used.

MAKE
: may contain command-line options. This is historical for BSD Make, and does not exist in the POSIX-conformant version of the utility.

MAKEFLAGS
: may contain anything that may be specified on *make*'s command line. Anything specified on *make*'s command line is appended to the *MAKEFLAGS* variable which is then entered into the environment for all programs which *make* executes.

  There are two possible formats for the string stored in *MAKEFLAGS*: The first is identical to the command-line (options preceded by hyphens and separated by blank characters). The second is useful only if there are no arguments to the options, and consists of option letters without leading hyphens or blank character separators.

  The difference between the command line and the *MAKEFLAGS* variable is that the contents of the environment variable are not expanded by the shell.

MAKEOBJDIR
: A path to the directory where the targets are built.

MAKESTARTUP
: A path to a startup file which is used instead of the system startup file, **$OPENNT_ROOT/usr/share/mk/sys.mk**.

PWD
: An alternate path to the current directory. If set, this value will be used to define the value of the *.CURDIR* variable.

FILES

Make uses the following files.

.depend
: List of dependencies; created by the *mkdep* utility (not included in the INTERIX Software Development Kit). If this file exists, it is read.

Makefile
: List of dependencies and recipes; this is read only if **makefile** does not exist in the current directory.

makefile
: List of dependencies and recipes; used in preference to **makefile**.

sys.mk
: System makefile; not read if **–r** is specified. The directory containing the system makefile can be specified with the **–I** option.

$OPENNT_ROOT/usr/share/mk
: System makefile directory. The root of the path depends upon where the INTERIX distribution is installed. The directory containing the system makefile can be specified with the **–I** option.

NAME

makedepend – create dependencies in makefiles

SYNOPSIS

```
makedepend [-Dname=def] [-Dname] [-Iincludedir]
           [-a] [-fmakefile] [-oobjsuffix]
           [-pobjprefix] [-sstring] [-wwidth]
           [- -otheroptions - -]sourcefile ...
```

DESCRIPTION

Makedepend reads each *sourcefile* in sequence and parses it like a C-preprocessor, processing all **#include**, **#define**, **#undef**, **#ifdef**, **#ifndef**, **#endif**, **#if** and **#else** directives so that it can correctly tell which **#include,** directives would be used in a compilation. Any **#include,** directives can reference files having other **#include** directives, and parsing will occur in these files as well.

Every file that a *sourcefile* includes, directly or indirectly, is what *makedepend* calls a "dependency". These dependencies are then written to a **makefile** in such a way that *make* will know which object files must be recompiled when a dependency has changed.

By default, *makedepend* places its output in the file named **makefile** if it exists, otherwise **Makefile.** An alternate makefile may be specified with the **–f** option. It first searches the makefile for the line

```
# DO NOT DELETE THIS LINE -- make depend depends on it.
```

or one provided with the **–s** option, as a delimiter for the dependency output. If it finds it, it will delete everything following this to the end of the makefile and put the output after this line. If it doesn't find it, the program will append the string to the end of the makefile and place the output following that. For each *sourcefile* appearing on the command line, *makedepend* puts lines in the makefile of the form

```
sourcefile.o: dfile ...
```

Where "sourcefile.o" is the name from the command line with its suffix replaced with ".o", and "dfile" is a dependency discovered in a **#include** directive while parsing *sourcefile* or one of the files it included.

EXAMPLE

Normally, *makedepend* will be used in a makefile target so that typing "make depend" will bring the dependencies up to date for the makefile. For example,

```
SRCS = file1.c file2.c ...
CFLAGS = -O -DHACK -I../foobar -xyz
depend:
        makedepend -- $(CFLAGS) -- $(SRCS)
```

OPTIONS

Makedepend will ignore any option that it does not understand so that you may use the same arguments that you would for *cc*.

–D *name*=*def*

–D *name*

Define. This places a definition for *name* in *makedepend*'s symbol table. Without =*def* the symbol becomes defined as "1".

–I*includedir*

Include directory. This option tells *makedepend* to prepend *includedir* to its list of directories to search when it encounters a **#include** directive. By default, *makedepend* only searches **$OPENNT_ROOT/usr/include**.

–a Append the dependencies to the end of the file instead of replacing them.

–f*makefile*

Filename. This allows you to specify an alternate makefile in which *makedepend* can place its output.

–o*objsuffix*

Object file suffix. Some systems may have object files whose suffix is something other than ".o". This option allows you to specify another suffix, such as ".b" with **-o.b** or ":obj" with **-o:obj** and so forth.

–p*objprefix*

Object file prefix. The prefix is prepended to the name of the object file. This is usually used to designate a different directory for the object file. The default is the empty string.

–s*string*

Starting string delimiter. This option permits you to specify a different string for *makedepend* to look for in the makefile.

–w *width*

Line width. Normally, *makedepend* will ensure that every output line that it writes will be no wider than 78 characters for the sake of readability. This option enables you to change this width.

–– *options* **––**

If *makedepend* encounters a double hyphen (––) in the argument list, then any unrecognized argument following it will be silently ignored; a second double hyphen terminates this special treatment. In this way, *makedepend* can be made to safely ignore esoteric compiler arguments that might normally be found in a CFLAGS *make* macro (see the "EXAMPLE" section above). All options that *makedepend* recognizes and appear between the pair of double hyphens are processed normally.

ALGORITHM

The approach used in this program enables it to run an order of magnitude faster than any other "dependency generator" I have ever seen. Central to this performance are two assumptions: that all files compiled by a single makefile will be compiled with roughly the same **–I** and **–D** options; and that most files in a single directory will include largely the same files.

Given these assumptions, *makedepend* expects to be called once for each makefile, with all source files that are maintained by the makefile appearing on

the command line. It parses each source and include file exactly once, maintaining an internal symbol table for each. Thus, the first file on the command line will take an amount of time proportional to the amount of time that a normal C preprocessor takes. But on subsequent files, if it encounters an include file that it has already parsed, it does not parse it again.

For example, imagine you are compiling two files, **file1.c** and **file2.c,** they each include the header file and the file **header.h** in turn includes the files **def1.h** and **def2.h.** When you run the command

```
makedepend file1.c file2.c
```

makedepend will parse **file1.c** and consequently, **header.h** and then **def1.h** and **def2.h.** It then decides that the dependencies for this file are

```
file1.o: header.h def1.h def2.h
```

But when the program parses **file2.c** and discovers that it, too, includes **header.h,** it does not parse the file, but simply adds **header.h, def1.h** and **def2.h** to the list of dependencies for **file2.o**.

SEE ALSO

cc, *make*

BUGS

If you do not have the source for *cpp*, the Berkeley C preprocessor, then *makedepend* will be compiled in such a way that all **#if** directives will evaluate to "false" regardless of their actual value. This may cause the wrong **#include** directives to be evaluated. *Makedepend* should simply have its own parser written for **#if** expressions.

Imagine you are parsing two files, say **file1.c** and **file2.c,** each includes the file **def.h.** The list of files that **def.h** includes might truly be different when **def.h** is included by **file1.c** than when it is included by **file2.c**. But once *makedepend* arrives at a list of dependencies for a file, it is cast in concrete.

AUTHOR

Todd Brunhoff, Tektronix, Inc. and MIT Project Athena

NAME

man – display system documentation

SYNOPSIS

```
man [–M manpaths] [–s section] name
man section name
man –k topic
```

DESCRIPTION

The *man* command displays information about the given command or function *name*. Where both a system interface call and a command have the same name (such as *printf* and *printf*( )), you can specify which you want by specifying a section name with **–s** *section* (or, in the second form, *section*).

The options are:

–M *manpath*

The *man* command searches the directories specified in *manpath* as well as the standard directories. *Manpath* is a colon-separated list of directories. The *manpath* is used *instead of* the contents of the *MANPATH* environment variable (if set).

–s *section*

Look for the man pages in the subdirectory named **cat***section*. By default, *man* searches through sections named 1 – 9, l (letter ell, for "local"), n (for "new"), and X (for X11 man pages).

Section 1 is utilities; section 2 is calls traditionally implemented as system calls; section 3 is library routines.

Section	Contents
1	Commands and utilities
2	"System calls"
3	Library routines
4	Devices and drivers
5	File formats and other miscellaneous information

The term "system calls" is quoted because these are calls traditionally associated with the kernel. Since the INTERIX API is not in the kernel itself but in the subsystem, they are not really system calls.

man displays the first matching reference page it finds.

The section numbers are used to distinguish between reference pages with the same names, such as *printf* and *printf*( ). To see the reference page for the *printf* utility, use the command:

```
$ man -s 1 printf
```

and to see the reference page for the *printf*( ) function, use:

```
$ man -s 3 printf
```

You can also use *man* to look for commands and calls related to a particular topic. The **–k** option searches for a particular keyword:

–k *topic* Search the **whatis.db** database for a reference page summary line that contains the keyword *topic*. The name of the reference page appears in parentheses. The *topic* can be an extended regular expression; the search is case-insensitive. (Many people prefer to alias **man -k** to the command *apropos*.)

The **whatis.db** database is simply the collection of utility and function descriptions found under the *NAME* headings of the reference pages.

EXAMPLES

To find all programs for editing files:

```
$ man -k edit
 ed - line oriented text editor
 ex, vi - Visual editor
 sed - stream editor
```

To see the reference page for the *printf*( ) function:

```
$ man 3 printf
```

DIAGNOSTICS

man exits with 0 for success and >0 for failure. It returns 1 if the *topic* is not found in any whatis database.

FILES

Man uses the following files and directories:

$OPENNT_ROOT/share/man/cat*section*
The directories containing the formatted reference pages.

$OPENNT_ROOT/share/man/whatis.db
The keyword database. This is a text file containing the command summary descriptions from the reference pages, one to a line. For reference pages which document more than one utility or function, parentheses surround the utility or function which designates the page.

ENVIRONMENT VARIABLES

Man makes use of the following environment variable, if set:

MANPATH
A colon-separated list of directories to search for man pages. Each directory is expected to contain a set of **cat***section* subdirectories and a **whatis.db** file.

PAGER A command to invoke the pager program that displays the reference page. The default is **more**.

SEE ALSO

grep, *more*

NAME

maze – an automated X11 demo repeatedly creating and solving a random maze

SYNOPSIS

`maze [-S] [-r] [-g geometry] [-d display]`

DESCRIPTION

The *maze* program creates a "random" maze and then solves it with graphical feedback.

Maze accepts the following options:

–S Full screen window option.

–r Reverse video option.

–g *geometry*
Specifies the window geometry to be used.

–d *display*
Specifies the display to be used.

The following lists the current functionality of various mouse button clicks;

LeftButton
Clears the window and restarts maze.

MiddleButton
Toggles the maze program, first click -> *stop*, second click -> *continue*.

RightButton
Kills maze.

LIMITATIONS

No color support.

Expose events force a restart of maze.

Currently, mouse actions are based on "raw" values [Button1, Button2 and Button3] from the ButtonPress event.

Doesn’t use pointer mapping.

COPYRIGHT

Copyright 1988 by Sun Microsystems, Inc. Mountain View, CA.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Sun or MIT not be used in advertising or publicity pertaining to distribution of the software without specific prior written permission. Sun and M.I.T. make no representations about the suitability of this software for any purpose. It is provided "as is" without any express or implied warranty.

SUN DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SUN BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

AUTHOR(s)

Richard Hess[X11 extensions] {...}!uunet!cimshop!rhess
Consilium, Mountain View, CA

Dave Lemke[X11 version]lemke@sun.COM
Sun MicroSystems, Mountain View, CA

Martin Weiss[SunView version]
Sun MicroSystems, Mountain View, CA

NAME

merge – three-way file merge

SYNOPSIS

`merge [-pq] [-L label1 [-L label3]] file1 file2 file3`

DESCRIPTION

merge incorporates all changes that lead from *file2* to *file3* into *file1*. The result goes to standard output if **–p** is present, into *file1* otherwise. *merge* is useful for combining separate changes to an original. Suppose *file2* is the original, and both *file1* and *file3* are modifications of *file2*. Then *merge* combines both changes.

An overlap occurs if both *file1* and *file3* have changes in a common segment of lines. On a few older hosts where *diff3* does not support the **–E** option, *merge* does not detect overlaps, and merely supplies the changed lines from *file3*. On most hosts, if overlaps occur, *merge* outputs a message (unless the **–q** option is given), and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If there are overlaps, the user should edit the result and delete one of the alternatives. If the **–L** *label1* and **–L** *label3* options are given, the labels are output in place of the names *file1* and *file3* in overlap reports.

DIAGNOSTICS

Exit status is 0 for no overlaps, 1 for some overlaps, 2 for trouble.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 1.5; Release Date: 1997/11/12.
Copyright © 1982, 1988, 1989 by Walter F. Tichy.
Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

diff3, *diff*, *rcsmerge*, *co*.

NAME

mkdir – make directories

SYNOPSIS

`mkdir [-p] [-m mode] directory_name ...`

DESCRIPTION

Mkdir creates the directories named as operands, in the order specified, using mode `rwxrwxrwx` (0777) as modified by the current *umask*.

The options are as follows:

–m Set the file permission bits of the final created directory to the specified *mode*. The *mode* argument can be in any of the formats specified to the *chmod* command. If a symbolic mode is specified, the operation characters "+" and "-" are interpreted relative to an initial mode of `a=rwx`.

–p Create intermediate directories as required. If this option is not specified, the full path prefix of each operand must already exist. Intermediate directories are created with permission bits of `rwxrwxrwx` (0777) as modified by the current umask, plus write and search permission for the owner.

The user must have write permission in the parent directory.

DIAGNOSTICS

Mkdir exits 0 if successful, and >0 if an error occurred.

SEE ALSO

rm, *rmdir*, *umask*

NAME

mkdirhier – makes a directory hierarchy

SYNOPSIS

`mkdirhier` *`directory`* `...`

DESCRIPTION

The *mkdirhier* command creates the specified directories. Unlike *mkdir* if any of the parent directories of the specified directory do not exist, it creates them as well.

SEE ALSO

mkdir

NAME

mkfifo – make fifos

SYNOPSIS

`mkfifo [-m` *`mode`*`]` *`fifo_name ...`*

DESCRIPTION

Mkfifo creates the fifos requested, in the order specified, using mode `rwxrwxrwx` (0777).

The options are as follows:

–m Set the file permission bits of the created fifo to the specified *mode*. The argument can be in any of the formats specified to the *chmod* command. If a symbolic mode is specified, the operation characters ‘‘+’’ and ‘‘-’’ are interpreted relative to an initial mode of `a=rwx`.

Mkfifo requires write permission in the parent directory.

DIAGNOSTICS

Mkfifo exits 0 if successful, and >0 if an error occurred.

SEE ALSO

mkdir, *rm*

NAME

mkfontdir - create fonts.dir file from directory of font files

SYNOPSIS

`mkfontdir [`*`directory-names`*`]`

DESCRIPTION

For each directory argument, *mkfontdir* reads all of the font files in the directory searching for properties named "FONT", or (failing that) the name of the file stripped of its suffix. These are used as font names, which are written out to the file "fonts.dir" in the directory along with the name of the font file.

The kinds of font files read by *mkfontdir* depends on configuration parameters, but typically include PCF (suffix ".pcf"), SNF (suffix ".snf") and BDF (suffix ".bdf"). If a font exists in multiple formats, *mkfontdir* will first choose PCF, then SNF and finally BDF.

SCALABLE FONTS

Because scalable font files do not usually include the X font name, the fonts.dir file in directories containing such fonts must be edited by hand to include the appropriate entries for those fonts. However, when *mkfontdir* is run, all of those additions will be lost, so be careful.

FONT NAME ALIASES

The file **fonts.alias** which can be put in any directory of the font-path is used to map new names to existing fonts, and should be edited by hand. The format is straight forward enough, two white-space separated columns, the first containing aliases and the second containing font-name patterns.

When a font alias is used, the name it references is search for in the normal manner, looking through each font directory in turn. This means that the aliases need not mention fonts in the same directory as the alias file.

To embed white-space in either name, simply enclose them in double-quote marks, to embed double-quote marks (or any other character), precede them with back-slash:

```
"magic-alias with spaces"  "\"font\name\" with quotes"
regular-alias              fixed
```

If the string "FILE_NAMES_ALIASES" stands alone on a line, each file-name in the directory (stripped of it's suffix) will be used as an alias for that font.

USAGE

Both the **X server** and the **Font Server** look for **fonts.dir** and **fonts.alias** files in each directory in the font path each time it is set (see *xset*).

SEE ALSO

X, *Xserver*, *xset*

NAME

more – display file on crt terminal

SYNOPSIS

`more [-ceinus] [-t tag] [-x tabs] [-/pattern] [-#] [file ... ]`

DESCRIPTION

More is a filter for paging through text one screenful at a time. It uses *termcap* so it can run on a variety of terminals. There is even limited support for hardcopy terminals. (On a hardcopy terminal, lines which should be printed at the top of the screen are prefixed with an up-arrow.) If *file* is a single hyphen (-), *more* reads standard input (though it won't take input from a terminal).

Command line options are described below. Options are also taken from the environment variable *MORE* but command line options will override them. Make sure to precede options with a dash (–) in the *MORE* variable.

–c Normally, *more* will repaint the screen by scrolling from the bottom of the screen. If the **–c** option is set, when *more* needs to change the entire display, it will paint from the top line down.

–e When standard output is a terminal, *more* normally prompts the user, indicating end-of-file, when it reaches the end of the last argument, and waits. When **–e** is specified, *more* exits as soon as it writes the last line of the last file.

–i The **–i** option causes searches to ignore case; that is, uppercase and lowercase are considered identical.

–n The **–n** flag suppresses line numbers. The default (to use line numbers) may cause *more* to run more slowly in some cases, especially with a very large input file. Suppressing line numbers with the **–n** flag will avoid this problem. Using line numbers means: the line number will be displayed in the **=** command, and the **v** command will pass the current line number to the editor.

–s The **–s** option causes consecutive blank lines to be squeezed into a single blank line.

–t The **–t** option, followed immediately by a tag, will edit the file containing that tag. For more information, see the *ctags* command.

–u By default, *more* treats backspaces and CR-LF sequences specially. Backspaces which appear adjacent to an underscore character are displayed as underlined text. Backspaces which appear between two identical characters are displayed as emboldened text. CR-LF sequences are compressed to a single linefeed character. The **–u** option causes backspaces to always be displayed as control characters, i.e. as the two character sequence ''^H'', and CR-LF to be left alone.

–x *N* The **–x** option sets tab stops every *N* positions. The default for *N* is 8.

–/*pattern* The **–/** option specifies a string that will be searched for before each file is displayed.

COMMANDS

Interactive commands for *more* are based on *vi*. Some commands may be preceded by a decimal number, called N in the descriptions below. In the following descriptions, ^X means control-X.

h Help: display a summary of these commands. If you forget all the other commands, remember this one.

<SPACE>, **f**, or **^F**
Scroll forward *N* lines, default one window. If *N* is more than the screen size, only the final screenful is displayed.

b or **^B** Scroll backward *N* lines, default one window (see option **–z** below). If *N* is more than the screen size, only the final screenful is displayed.

j or <RETURN>
Scroll forward *N* lines, default 1. The entire *N* lines are displayed, even if *N* is more than the screen size.

k Scroll backward *N* lines, default 1. The entire *N* lines are displayed, even if *N* is more than the screen size.

d or **^D** Scroll forward *N* lines, default one half of the screen size. If *N* is specified, it becomes the new default for subsequent d and u commands.

u or **^U** Scroll backward *N* lines, default one half of the screen size. If *N* is specified, it becomes the new default for subsequent d and u commands.

g Go to line *N* in the file, default 1 (beginning of file).

G Go to line *N* in the file, default the end of the file.

p or **%** Go to a position *N* percent into the file. *N* should be between 0 and 100. (This works if standard input is being read, but only if *more* has already read to the end of the file. It is always fast, but not always useful.)

r or **^L** Repaint the screen.

R Repaint the screen, discarding any buffered input. Useful if the file is changing while it is being viewed.

m*c* Followed by any lowercase letter, *c*, marks the current position with that letter.

'*c* (Single quote.) Followed by any lowercase letter, returns to the position which was previously marked with that letter. Followed by another single quote, returns to the position at which the last "large" movement command was executed, or the beginning of the file if no such movements have occurred. All marks are lost when a new file is examined.

/pattern Search forward in the file for the *N*-th line containing the pattern. *N* defaults to 1. The pattern is a regular expression, as recognized by *ed*. The search starts at the second line displayed.

?*pattern* Search backward in the file for the *N*-th line containing the pattern. The search starts at the line immediately before the top line displayed.

/! *pattern*
Like **/**, but the search is for the *N*-th line which does *not* contain the pattern.

?!*pattern*

Like **?**, but the search is for the *N*-th line which does *not* contain the pattern.

n Repeat previous search, for *N*-th line containing the last pattern (or *not* containing the last pattern, if the previous search was **/!** or **?!**).

E[*filename*]
Examine a new file. If the filename is missing, the "current" file (see the *N* and P commands below) from the list of files in the command line is re-examined. If the filename is a pound sign (#), the previously examined file is re-examined.

N or **:n** Examine the next file (from the list of files given in the command line). If a number *N* is specified (not to be confused with the command **N**), the *N*-th next file is examined.

P or **:p** Examine the previous file. If a number *N* is specified, the *N*-th previous file is examined.

:t *tag* Go to supplied tag.

v Invokes an editor to edit the current file being viewed. The editor is taken from the environment variable *EDITOR*, or defaults to *vi*.

= or **^G** These options print out the number of the file currently being displayed relative to the total number of files there are to display, the current line number, the current byte number and the total bytes to display, and what percentage of the file has been displayed. If *more* is reading from stdin, or the file is shorter than a single screen, some of these items may not be available. Note, all of these items reference the first byte of the last line displayed on the screen.

q or **:q** or **ZZ**
Exits *more*.

ENVIRONMENT

More uses the following environment variables, if they are set:

MORE This variable may be set with favored options to *more*.

EDITOR
Specify default editor.

SHELL Current shell in use (normally set by the shell at login time).

TERM Specifies terminal type, used by more to get the terminal characteristics necessary to manipulate the screen.

SEE ALSO

ctags, *vi*

AUTHOR

This software is derived from software contributed to Berkeley by Mark Nudleman.

NAME

mt – magnetic tape utility

SYNOPSIS

`mt [-hv] [-f tapename] [command args...]`

DESCRIPTION

Mt is used to give commands to a magnetic tape drive.

If the **–h** option is given, a description of the command line argument syntax is written to standard output, and the program exits without opening the tape drive.

If the **–f** option is given, then *tapename* specifies which tape drive is to be operated upon. *Tapename* must have the form **/dev/ntape***N* or **/dev/tape***N*, where *N* is a decimal digit. If the **–f** option is not specified, then the tapename is obtained from the *TAPE* environment variable, and if that isn't set, the default tapename is **/dev/tape0**.

If the **–v** option is given, *mt* prints a verbose description of what it is doing to standard output. If you want to know why *mt* takes so long to perform some operations on your tape drive, use **–v** to find out what it is doing.

Following the command line options is a command name, which may be followed by arguments.

The following commands are supported:

online Bring the tape unit on-line.

read *file* Read the tape until a tape mark or end-of-file is reached, and store the resulting data in *file*. The blocksize used for reading the tape is the current blocksize, as set by

```
mt setblk
```

The command fails if the current blocksize is 0. The tape is not rewound after the file is read; the tape is left positioned at the tape mark or end-of-file.

The command **read nul** will read the tape and discard any information read, so the result is to forward space to the next tape mark.

rewind Rewind the tape.

setblk *count*

Set the block size of the drive to *count* bytes per record. A *count* of 0 means variable length blocks. Use

```
mt status
```

to determine the minimum, maximum and default block size (this is part of the 'tape drive' information), and the current block size (this is part of the 'tape media' information).

status Print tape drive and tape media parameters to stdout.

DIAGNOSTICS

The *mt* utility has an exit status of 0 on success, 1 on a syntax error interpreting command line arguments, and 2 on a tape error.

SEE ALSO

dd

NAME

mv – move files

SYNOPSIS

```
mv [–fi] source target
mv [–fi] source ... directory
```

DESCRIPTION

In its first form, the *mv* utility renames the file named by the *source* operand to the destination path named by the *target* operand. This form is assumed when the last operand does not name an already existing directory.

In its second form, *mv* moves each file named by a *source* operand to a destination file in the existing directory named by the *directory* operand. The destination path for each operand is the pathname produced by the concatenation of the last operand, a slash, and the final pathname component of the named file.

The following options are available:

–f Do not prompt for confirmation before overwriting the destination path. (The **–i** option is ignored if the **–f** option is specified.)

–i Causes *mv* to write a prompt to standard error before moving a file that would overwrite an existing file. If the response from the standard input begins with the character ‘‘y’’, the move is attempted.

It is an error for either the *source* operand or the destination path to specify a directory unless both do.

If the destination path does not have a mode which permits writing, *mv* prompts the user for confirmation as specified for the **–i** option.

As the *rename*( ) call does not work across file systems, *mv* uses *cp* and *rm* to accomplish the move. The effect is equivalent to:

```
rm -f destination_path && \
cp -pr source_file destination && \
rm -rf source_file
```

The *mv* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

cp, *ln*

NAME

mvwtmpx – rename current wtmpx file and start new one

SYNOPSIS

`mvwtmpx` *newname*

DESCRIPTION

The *mvwtmpx* utility renames the current **$OPENNT_ROOT/var/adm/wtmpx** file to *newname* and ensures that a new one is started in its place. No data will be lost during the renaming process.

DIAGNOSTICS

The *mvwtmpx* utility exits with status 0 for success, and >0 if an error occurred.

FILES

$OPENNT_ROOT/var/adm/wtmpx
The current **wtmpx** file.

SEE ALSO

renamewtmpx()

NAME

mwm – The Motif Window Manager

SYNOPSIS

mwm [*options*]

DESCRIPTION

mwm is an X Window System client that provides window management functionality and some session management functionality. It provides functions that facilitate control (by the user and the programmer) of elements of window state such as placement, size, icon / normal display, and input-focus ownership. It also provides session management functions such as stopping a client.

Options

–display *display*
This option specifies the display to use; see *X(1)*.

–xrm *resourcestring*
This option specifies a resource string to use.

–multiscreen
This option causes *mwm* to manage all screens on the display. The default is to manage only a single screen.

–name *name*
This option causes *mwm* to retrieve its resources using the specified name, as in *name*resource*.

–screens *name* [*name* [...]]
This option specifies the resource names to use for the screens managed by *mwm*. If *mwm* is managing a single screen, only the first name in the list is used. If *mwm* is managing multiple screens, the names are assigned to the screens in order, starting with screen 0. Screen 0 gets the first name, screen 1 the second name, and so on.

Appearance

The following sections describe the basic default behaviors of windows, icons, the icon box, input focus, and window stacking. The appearance and behavior of the window manager can be altered by changing the configuration of specific resources. Resources are defined under the heading "X DEFAULTS."

Screens

By default, *mwm* manages only the single screen specified by the **–display** option or the DISPLAY environment variable (by default, screen 0). If the **–multiscreen** option is specified or if the **multiScreen** resource is True, *mwm* tries to manage all the screens on the display.

When *mwm* is managing multiple screens, the **–screens** option can be used to give each screen a unique resource name. The names are separated by blanks, for example, **–screens** mwm0 mwm1. If there are more screens than names, resources for the remaining screens will be retrieved using the first name. By

default, the screen number is used for the screen name.

Windows

Default *mwm* window frames have distinct components with associated functions:

Title Area In addition to displaying the client's title, the title area is used to move the window. To move the window, place the pointer over the title area, press button 1 and drag the window to a new location. By default, a wire frame is moved during the drag to indicate the new location. When the button is released, the window is moved to the new location.

Title Bar The title bar includes the title area, the minimize button, the maximize button, and the window menu button. In shaped windows, such as round windows, the title bar floats above the window.

Minimize Button

To turn the window into an icon, click button 1 on the minimize button (the frame box with a *small* square in it).

Maximize Button

To make the window fill the screen (or enlarge to the largest size allowed by the configuration files), click button 1 on the maximize button (the frame box with a *large* square in it).

Window Menu Button

The window menu button is the frame box with a horizontal bar in it. To pull down the window menu, press button 1. While pressing, drag the pointer on the menu to your selection, then release the button when your selection is highlighted. Pressing button 3 in the title bar or resize border handles also posts the window menu. Alternately, you can click button 1 to pull down the menu and keep it posted; then position the pointer and select. You can also post the window menu by pressing Shift Esc or Alt Double-clicking button 1 with the pointer on the window menu button closes the window. The following table lists the contents of the window menu.

Default Window Menu

Selection	Accelerator	Description
Restore	Alt+F5	Restores the window to its size before minimizing or maximizing
Move	Alt+F7	Allows the window to be moved with keys or mouse
Size	Alt+F8	Allows the window to be resized
Minimize	Alt+F9	Turns the window into an icon
Maximize	Alt+F10	Makes the window fill the screen
Lower	Alt+F3	Moves window to bottom of window stack
Close	Alt+F4	Causes client to terminate

Resize Border Handles

To change the size of a window, move the pointer over a resize border handle (the cursor changes), press button 1, and drag the window to a new size. When the button is released, the window is resized. While dragging is being done, a rubber-band outline is displayed to indicate the new window size.

Matte

An optional matte decoration can be added between the client area and the window frame. A matte is not actually part of the window frame. There is no functionality associated with a matte.

Icons

Icons are small graphic representations of windows. A window can be minimized (iconified) using the minimize button on the window frame. Icons provide a way to reduce clutter on the screen.

Pressing mouse button 1 when the pointer is over an icon causes the icon's window menu to pop up. Releasing the button (press + release without moving mouse = click) causes the menu to stay posted. The menu contains the following selections:

Icon Window Menu

Selection	Accelerator	Description
Restore	Alt+F5	Opens the associated window
Move	Alt+F7	Allows the icon to be moved with keys
Size	Alt+F8	Inactive (not an option for icons)
Minimize	Alt+F9	Inactive (not an option for icons)
Maximize	Alt+F10	Opens the associated window and makes it fill the screen
Lower	Alt+F3	Moves icon to bottom of icon stack
Close	Alt+F4	Removes client from *mwm* management

Note that pressing button 3 over an icon also causes the icon's window menu to pop up. To make a menu selection, drag the pointer over the menu and release button 3 when the desired item is highlighted.

Double-clicking button 1 on an icon invokes the **f.restore_and_raise** function and restores the icon's associated window to its previous state. For example, if a maximized window is iconified, then double-clicking button 1 restores it to its maximized state. Double-clicking button 1 on the icon box's icon opens the icon box and allows access to the contained icons. (In general, double-clicking a mouse button is a quick way to perform a function.) Pressing Shift Esc or Menu (the pop-up menu key) causes the icon window menu of the currently selected icon to pop up.

Icon Box

When icons begin to clutter the screen, they can be packed into an icon box. (To use an icon box, *mwm* must be started with the icon box configuration already set.) The icon box is a *mwm* window that holds client icons. It includes one or more scroll bars when there are more window icons than the icon box can show at the same time.

Icons in the icon box can be manipulated with the mouse. The following table summarizes the behavior of this interface. Button actions apply whenever the pointer is on any part of the icon. Note that double-clicking an icon in the icon box invokes the **f.restore_and_raise** function.

Button Action	**Description**
Button 1 click	Selects the icon
Button 1 double-click	Normalizes (opens) the associated window Raises an already open window to the top of the stack
Button 1 drag	Moves the icon
Button 3 press	Causes the menu for that icon to pop up
Button 3 drag	Highlights items as the pointer moves across the menu

Pressing mouse button 3 when the pointer is over an icon causes the menu for that icon to pop up.

Icon Menu for the Icon Box

Selection	Accelerator	Description
Restore	Alt+F5	Opens the associated window (if not already open)
Move	Alt+F7	Allows the icon to be moved with keys
Size	Alt+F8	Inactive
Minimize	Alt+F9	Inactive
Maximize	Alt+F10	Opens the associated window (if not already open) and maximizes its size
Lower	Alt+F3	Inactive
Close	Alt+F4	Removes client from *mwm* management

To pull down the window menu for the icon box itself, press button 1 with the pointer over the menu button for the icon box. The window menu of the icon box differs from the window menu of a client window: The "Close" selection is replaced with the "PackIcons Shift+Alt+F7" selection. When selected, PackIcons packs the icons in the box to achieve neat rows with no empty slots.

You can also post the window menu by pressing Shift Esc or Alt-Space. Pressing Menu (the pop-up menu key) causes the icon window menu of the currently selected icon to pop up.

Input Focus

mwm supports (by default) a keyboard input focus policy of explicit selection. This means when a window is selected to get keyboard input, it continues to get keyboard input until the window is withdrawn from window management, another window is explicitly selected to get keyboard input, or the window is iconified. Several resources control the input focus. The client window with the keyboard input focus has the active window appearance with a visually distinct window frame.

The following tables summarize the keyboard input focus selection behavior:

Button Action	Object	Function Description
Button 1 press	Window / window frame	Keyboard focus selection
Button 1 press	Icon	Keyboard focus selection

Key Action	Function Description
[Alt][Tab]	Move input focus to next window in window stack (available only in explicit focus mode)
[Alt][Shift][Tab]	Move input focus to previous window in window stack (available only in explicit focus mode)

Window Stacking

There are two types of window stacks: global window stacks and an application's local family window stack.

The global stacking order of windows may be changed as a result of setting the keyboard input focus, iconifying a window, or performing a window manager window stacking function. When keyboard focus policy is explicit the default value of the **focusAutoRaise** resource is True. This causes a window to be raised to the top of the stack when it receives input focus, for example, by pressing button 1 on the title bar. The key actions defined in the previous table will thus raise the window receiving focus to the top of the stack.

In pointer mode, the default value of **focusAutoRaise** is False, that is, the window stacking order is not changed when a window receives keyboard input focus. The following key actions can be used to cycle through the global window stack.

Key Action	**Function Description**
[Alt][ESC]	Place top window on bottom of stack
[Alt][Shift][ESC]	Place bottom window on top of stack

By default, a window's icon is placed on the bottom of the stack when the window is iconified; however, the default can be changed by the **lowerOnIconify** resource.

Transient windows (secondary windows such a dialog boxes) stay above their parent windows by default. However, an application's local family stacking order may be changed to allow a transient window to be placed below its parent top-level window. The following parameters show the modification of the stacking order for the **f.lower** function.

f.lower Lowers the transient window within the family (staying above the parent) and lowers the family in the global window stack.

f.lower [**within**]
Lowers the transient window within the family (staying above the parent) but does not lower the family in the global window stack.

f.lower [**freeFamily**]
Lowers the window free from its family stack (below the parent), but does not lower the family in the global window stack.

The parameters **within** and **freeFamily** can also be used with **f.raise** and **f.raise_lower**.

X Defaults

mwm is configured from its resource database. This database is built from the following sources. They are listed in order of precedence, low to high:

`/usr/lib/X11/app-defaults/Mwm`
`$HOME/Mwm`
`RESOURCE_MANAGER root window property or $HOME/.Xdefaults`
`XENVIRONMENT variable or $HOME/.Xdefaults-`*host*
mwm command line options

The file names **/usr/lib/X11/app-defaults/Mwm** and **$HOME/Mwm** represent customary locations for these files. The actual location of the system-wide class resource file may depend on the XFILESEARCHPATH environment variable and the current language environment. The actual location of the user-specific class resource file may depend on the XUSERFILESEARCHPATH and XAPPLRESDIR environment variables and the current language environment.

Entries in the resource database may refer to other resource files for specific types of resources. These include files that contain bitmaps, fonts, and **mwm** specific resources such as menus and behavior specifications (for example, button and key bindings).

Mwm is the resource class name of **mwm** and *mwm* is the default resource name used by *mwm* to look up resources. the **–screens** command line option specifies resource names, such as "mwm_b+w" and "mwm_color".) In the following discussion of resource specification, "Mwm" and "mwm" (and the aliased *mwm* resource names) can be used interchangeably, but "mwm" takes precedence over "Mwm".

mwm uses the following types of resources:

Component Appearance Resources:

These resources specify appearance attributes of window manager user interface components. They can be applied to the appearance of window manager menus, feedback windows (for example, the window reconfiguration feedback window), client window frames, and icons.

General Appearance and Behavior Resources:

These resources specify *mwm* appearance and behavior (for example, window management policies). They are not set separately for different *mwm* user interface components.

Client Specific Resources:

These *mwm* resources can be set for a particular client window or class of client windows. They specify client-specific icon and client window frame appearance and behavior.

Resource identifiers can be either a resource name (for example, foreground) or a resource class (for example, Foreground). If the value of a resource is a filename and if the filename is prefixed by "~/", then it is relative to the path contained in the HOME environment variable (generally the user's home directory).

Component Appearance Resources

The syntax for specifying component appearance resources that apply to window manager icons, menus, and client window frames is

`Mwm*`*`resource_id`*

For example, **Mwm*foreground** is used to specify the foreground color for *mwm* menus, icons, client window frames, and feedback dialogs.

The syntax for specifying component appearance resources that apply to a particular *mwm* component is

`Mwm*[menu|icon|client|feedback]*`*`resource_id`*

If *menu* is specified, the resource is applied only to **mwm** menus; if *icon* is specified, the resource is applied to icons; and if *client* is specified, the resource is applied to client window frames. For example, **Mwm*icon*foreground** is used to specify the foreground color for *mwm* icons, **Mwm*menu*foreground** specifies the foreground color for *mwm* menus, and **Mwm*client*foreground** is used to specify the foreground color for *mwm* client window frames.

The appearance of the title area of a client window frame (including window management buttons) can be separately configured. The syntax for configuring the title area of a client window frame is

`Mwm*client*title*`*`resource_id`*

For example, **Mwm*client*title*foreground** specifies the foreground color for the title area. Defaults for title area resources are based on the values of the corresponding client window frame resources.

The appearance of menus can be configured based on the name of the menu. The syntax for specifying menu appearance by name is

`Mwm*menu*`*`menu_name`*`*`*`resource_id`*

For example, **Mwm*menu*my_menu*foreground** specifies the foreground color for the menu named **my_menu**. The user can also specify resources for window manager menu components, that is, the gadgets which comprise the menu. These may include for example, a menu title, title separator, one or more buttons, and separators. If a menu contains more than one instance of a class, such as multiple PushButtonGadgets, the name of the first instance is "PushButtonGadget1", the second is "PushButtonGadget2", and so on. The following list identifies the naming convention used for window manager menu components:

Menu Title LabelGadget	"TitleName"
Menu Title SeparatorGadget	"TitleSeparator"
CascadeButtonGadget	"CascadeButtonGadget<n>"
PushButtonGadget	"PushButtonGadget<n>"
SeparatorGadget	"SeparatorGadget<n>"

Refer to the man page for each class for a list of resources which can be specified.

The following component appearance resources that apply to all window manager parts can be specified:

Component Appearance Resources—All Window Manager Parts

Name	Class	Value Type	Default
background	Background	color	varies†
backgroundPixmap	BackgroundPixmap	string††	varies†
bottomShadowColor	Foreground	color	varies†
bottomShadowPixmap	BottomShadowPixmap	string††	varies†
fontList	FontList	string†††	"fixed"
foreground	Foreground	color	varies†
saveUnder	SaveUnder	T/F	F
topShadowColor	Background	color	varies†
topShadowPixmap	TopShadowPixmap	string††	varies†

†The default is chosen based on the visual type of the screen.
††Image name. See *XmInstallImage*( ).
†††X11 X Logical Font Description

background (class **Background**)
This resource specifies the background color. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

backgroundPixmap (class **BackgroundPixmap**)
This resource specifies the background Pixmap of the *mwm* decoration when the window is inactive (does not have the keyboard focus). The default value is chosen based on the visual type of the screen.

bottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color. This color is used for the lower and right bevels of the window manager decoration. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

bottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap. This Pixmap is used for the lower and right bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

fontList (class **FontList**)
This resource specifies the font used in the window manager decoration. The character encoding of the font should match the character encoding of the strings that are used. The default is "fixed."

foreground (class **Foreground**)
This resource specifies the foreground color. The default is chosen

based on the visual type of the screen.

saveUnder (class **SaveUnder**)
This is used to indicate whether "save unders" are used for *mwm* components. For this to have any effect, save unders must be implemented by the X server. If save unders are implemented, the X server saves the contents of windows obscured by windows that have the save under attribute set. If the saveUnder resource is True, *mwm* will set the save under attribute on the window manager frame of any client that has it set. If saveUnder is False, save unders will not be used on any window manager frames. The default value is False.

topShadowColor (class **Background**)
This resource specifies the top shadow color. This color is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

topShadowPixmap (class **TopShadowPixmap)**
This resource specifies the top shadow Pixmap. This Pixmap is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

The following component appearance resources that apply to frame and icons can be specified:

Frame and Icon Components

Name	Class	Value Type	Default
activeBackground	Background	color	varies†
activeBackgroundPixmap	BackgroundPixmap	string††	varies†
activeBottomShadowColor	Foreground	color	varies†
activeBottomShadowPixmap	BottomShadowPixmap	string††	varies†
activeForeground	Foreground	color	varies†
activeTopShadowColor	Background	color	varies†
activeTopShadowPixmap	TopShadowPixmap	string††	varies†

†The default is chosen based on the visual type of the screen.
††See **XmInstallImage(3X)**.

activeBackground (class **Background**)
This resource specifies the background color of the *mwm* decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBackgroundPixmap (class **ActiveBackgroundPixmap**)
This resource specifies the background Pixmap of the *mwm* decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the *mwm* decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap of the *mwm* decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeForeground (class **Foreground**)

This resource specifies the foreground color of the *mwm* decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowColor (class **Background**)
This resource specifies the top shadow color of the *mwm* decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap of the *mwm* decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

General Appearance and Behavior Resources

The syntax for specifying general appearance and behavior resources is

`Mwm*`*`resource_id`*

For example, **Mwm*keyboardFocusPolicy** specifies the window manager policy for setting the keyboard focus to a particular client window.

The following general appearance and behavior resources can be specified:

General Appearance and Behavior Resources

Name	Class	Value Type	Default
autoKeyFocus	AutoKeyFocus	T/F	T
autoRaiseDelay	AutoRaiseDelay	millisec	500
bitmapDirectory	BitmapDirectory	directory	/usr/include/\X11/bitmaps
buttonBindings	ButtonBindings	string	"DefaultBut\tonBindings"
cleanText	CleanText	T/F	T
clientAutoPlace	ClientAutoPlace	T/F	T
colormapFocusPolicy	ColormapFocusPolicy	string	keyboard
configFile	ConfigFile	file	.mwmrc
deiconifyKeyFocus	DeiconifyKeyFocus	T/F	T
doubleClickTime	DoubleClickTime	millisec.	multi-click time
enableWarp	enableWarp	T/F	T
enforceKeyFocus	EnforceKeyFocus	T/F	T
fadeNormalIcon	FadeNormalIcon	T/F	F
feedbackGeometry	FeedbackGeometry	string	center on screen
frameBorderWidth	FrameBorderWidth	pixels	varies
iconAutoPlace	IconAutoPlace	T/F	T
iconBoxGeometry	IconBoxGeometry	string	6x1+0-0
iconBoxName	IconBoxName	string	iconbox
iconBoxSBDisplayPolicy	IconBoxSBDisplayPolicy	string	all
iconBoxTitle	IconBoxTitle	XmString	Icons
iconClick	IconClick	T/F	T
iconDecoration	IconDecoration	string	varies

iconImageMaximum	IconImageMaximum	wxh	50x50
iconImageMinimum	IconImageMinimum	wxh	16x16
iconPlacement	IconPlacement	string	left bottom
iconPlacementMargin	IconPlacementMargin	pixels	varies
interactivePlacement	InteractivePlacement	T/F	F
keyBindings	KeyBindings	string	"DefaultKey\ Bindings"
keyboardFocusPolicy	KeyboardFocusPolicy	string	explicit
limitResize	LimitResize	T/F	T
lowerOnIconify	LowerOnIconify	T/F	T
maximumMaximumSize	MaximumMaximumSize	wxh (pixels)	2X screen w&h
moveThreshold	MoveThreshold	pixels	4
moveOpaque	MoveOpaque	T/F	F
multiScreen	MultiScreen	T/F	F
passButtons	PassButtons	T/F	F
passSelectButton	PassSelectButton	T/F	T
positionIsFrame	PositionIsFrame	T/F	T
positionOnScreen	PositionOnScreen	T/F	T
quitTimeout	QuitTimeout	millisec.	1000
raiseKeyFocus	RaiseKeyFocus	T/F	F
resizeBorderWidth	ResizeBorderWidth	pixels	varies
resizeCursors	ResizeCursors	T/F	T
screens	Screens	string	varies
showFeedback	ShowFeedback	string	all
startupKeyFocus	StartupKeyFocus	T/F	T
transientDecoration	TransientDecoration	string	menu title
transientFunctions	TransientFunctions	string	–minimize –maximize
useIconBox	UseIconBox	T/F	F
wMenuButtonClick	WMenuButtonClick	T/F	T
wMenuButtonClick2	WMenuButtonClick2	T/F	T

autoKeyFocus (class **AutoKeyFocus**)

This resource is available only when the keyboard input focus policy is explicit. If autoKeyFocus is given a value of True, then when a window with the keyboard input focus is withdrawn from window management or is iconified, the focus is set to the previous window that had the focus. If the value given is False, there is no automatic setting of the keyboard input focus. It is recommended that both **autoKeyFocus** and **startupKeyFocus** be True to work with tear off menus. The default value is True.

autoRaiseDelay (class **AutoRaiseDelay**)

This resource is available only when the focusAutoRaise resource is True and the keyboard focus policy is pointer. The autoRaiseDelay resource specifies the amount of time (in milliseconds) that **mwm** will wait before raising a window after it gets the keyboard focus. The default value of this resource is 500 (ms).

bitmapDirectory (class **BitmapDirectory**)

This resource identifies a directory to be searched for bitmaps referenced by *mwm* resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is /usr/include/X11/bitmaps. The directory

/usr/include/X11/bitmaps represents the customary locations for this directory. The actual location of this directory may vary on some systems. If the bitmap is not found in the specified directory, XBMLANGPATH is searched.

buttonBindings (class **ButtonBindings**)

This resource identifies the set of button bindings for window management functions. The named set of button bindings is specified in the *mwm* resource description file. These button bindings are *merged* with the built-in default bindings. The default value for this resource is "DefaultButtonBindings".

cleanText (class **CleanText**)

This resource controls the display of window manager text in the client title and feedback windows. If the default value of True is used, the text is drawn with a clear (no stipple) background. This makes text easier to read on monochrome systems where a backgroundPixmap is specified. Only the stippling in the area immediately around the text is cleared. If False, the text is drawn directly on top of the existing background.

clientAutoPlace (class **ClientAutoPlace**)

This resource determines the position of a window when the window has not been given a program- or user-specified position. With a value of True, windows are positioned with the top left corners of the frames offset horizontally and vertically. A value of False causes the currently configured position of the window to be used. In either case, *mwm* will attempt to place the windows totally on-screen. The default value is True.

colormapFocusPolicy (class **ColormapFocusPolicy**)

This resource indicates the colormap focus policy that is to be used. If the resource value is explicit, a colormap selection action is done on a client window to set the colormap focus to that window. If the value is pointer, the client window containing the pointer has the colormap focus. If the value is keyboard, the client window that has the keyboard input focus has the colormap focus. The default value for this resource is keyboard.

configFile (class **ConfigFile**)

The resource value is the pathname for an **mwm** resource description file.

If the pathname begins with "~/", *mwm* considers it to be relative to the user's home directory (as specified by the HOME environment variable). If the LANG environment variable is set, *mwm* looks for $HOME/$LANG/*configFile*. If that file does not exist or if LANG is not set, *mwm* looks for $HOME/*configFile*.

If the **configFile** pathname does not begin with ~/, *mwm* considers it to be relative to the current working directory.

If the **configFile** resource is not specified or if that file does not exist, *mwm* uses several default paths to find a configuration file. If the LANG environment variable is set, *mwm* looks for the configuration file first in $HOME/$LANG/.mwmrc. If that file does not exist or if LANG is not set, *mwm* looks for $HOME/.mwmrc. If that file does not exist and if LANG is set, *mwm* next looks for the file

system.mwmrc in the $LANG subdirectory of an implementation-dependent directory. (The default for this directory, if not changed by the implementation, is /usr/lib/X11.) If that file does not exist or if LANG is not set, *mwm* looks for the file system.mwmrc in the same implementation-dependent directory.

deiconifyKeyFocus (class **DeiconifyKeyFocus**)
This resource applies only when the keyboard input focus policy is explicit. If a value of True is used, a window receives the keyboard input focus when it is normalized (deiconified). True is the default value.

doubleClickTime (class **DoubleClickTime**)
This resource is used to set the maximum time (in ms) between the clicks (button presses) that make up a double-click. The default value of this resource is the display's multi-click time.

enableWarp (class **EnableWarp**)
The default value of this resource, True, causes *mwm* to warp the pointer to the center of the selected window during keyboard-controlled resize and move operations. Setting the value to False causes *mwm* to leave the pointer at its original place on the screen, unless the user explicitly moves it with the cursor keys or pointing device.

enforceKeyFocus (class **EnforceKeyFocus**)
If this resource is given a value of True, the keyboard input focus is always explicitly set to selected windows even if there is an indication that they are "globally active" input windows. (An example of a globally active window is a scroll bar that can be operated without setting the focus to that client.) If the resource is False, the keyboard input focus is not explicitly set to globally active windows. The default value is True.

fadeNormalIcon (class **FadeNormalIcon**)
If this resource is given a value of True, an icon is grayed out whenever it has been normalized (its window has been opened). The default value is False.

feedbackGeometry (class **FeedbackGeometry**)
This resource sets the position of the move and resize feedback window. If this resource is not specified, the default is to place the feedback window at the center of the screen. The value of the resource is a standard window geometry string with the following syntax:

[=]{+-}*xoffset***{+-}***yoffset***]**

frameBorderWidth (class **FrameBorderWidth**)
This resource specifies the width (in pixels) of a client window frame border without resize handles. The border width includes the 3-D shadows. The default value is based on the size and resolution of the screen.

iconAutoPlace (class **IconAutoPlace**)
This resource indicates whether the window manager arranges icons in a particular area of the screen or places each icon where the window was when it was iconified. The value True indicates that icons are arranged in a particular area of the screen, determined by the

iconPlacement resource. The value False indicates that an icon is placed at the location of the window when it is iconified. The default is True.

iconBoxGeometry (class **IconBoxGeometry**)
This resource indicates the initial position and size of the icon box. The value of the resource is a standard window geometry string with the following syntax:

[=**][***width*x*height***][{**+-**}***xoffset***{**+-**}***yoffset***]**

If the offsets are not provided, the iconPlacement policy is used to determine the initial placement. The units for width and height are columns and rows.

The actual screen size of the icon box window depends on the iconImageMaximum (size) and iconDecoration resources. The default value for size is (6 * iconWidth + padding) wide by (1 * iconHeight + padding) high. The default value of the location is +0 -0.

iconBoxName (class **IconBoxName**)
This resource specifies the name that is used to look up icon box resources. The default name is "iconbox".

iconBoxSBDisplayPolicy (class **IconBoxSBDisplayPolicy**)
This resource specifies the scroll bar display policy of the window manager in the icon box. The resource has three possible values: all, vertical, and horizontal. The default value, "all", causes both vertical and horizontal scroll bars always to appear. The value "vertical" causes a single vertical scroll bar to appear in the icon box and sets the orientation of the icon box to horizontal (regardless of the iconBoxGeometry specification). The value "horizontal" causes a single horizontal scroll bar to appear in the icon box and sets the orientation of the icon box to vertical (regardless of the iconBoxGeometry specification).

iconBoxTitle (class **IconBoxTitle**)
This resource specifies the name that is used in the title area of the icon box frame. The default value is "Icons".

iconClick (class **IconClick**)
When this resource is given the value of True, the system menu is posted and left posted when an icon is clicked. The default value is True.

iconDecoration (class **IconDecoration**)
This resource specifies the general icon decoration. The resource value is label (only the label part is displayed) or image (only the image part is displayed) or label image (both the label and image parts are displayed). A value of activelabel can also be specified to get a label (not truncated to the width of the icon) when the icon is selected. The default icon decoration for icon box icons is that each icon has a label part and an image part (label image). The default icon decoration for stand alone icons is that each icon has an active label part, a label part, and an image part (activelabel label image).

iconImageMaximum (class **IconImageMaximum**)
This resource specifies the maximum size of the icon *image*. The resource value is *width***x***height* (for example, 64x64). The maximum supported size is 128x128. The default value of this resource is 50x50.

iconImageMinimum (class **IconImageMinimum**)
This resource specifies the minimum size of the icon *image*. The resource value is *width***x***height* (for example, 32x50). The minimum supported size is 16x16. The default value of this resource is 16x16.

iconPlacement (class **IconPlacement**)
This resource specifies the icon placement scheme to be used. The resource value has the following syntax:

primary_layout secondary_layout [tight]

The layout values are one of the following:

Value	**Description**
top	Lay the icons out top to bottom.
bottom	Lay the icons out bottom to top.
left	Lay the icons out left to right.
right	Lay the icons out right to left.

A horizontal (vertical) layout value should not be used for both the *primary_layout* and the *secondary_layout* (for example, don't use top for the *primary_layout* and bottom for the *secondary_layout*). The *primary_layout* indicates whether, when an icon placement is done, the icon is placed in a row or a column and the direction of placement. The *secondary_layout* indicates where to place new rows or columns. For example, top right indicates that icons should be placed top to bottom on the screen and that columns should be added from right to left on the screen. The default placement is left bottom (icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows added from the bottom of the screen to the top of the screen). A *tight* value places icons with zero spacing in between icons. This value is useful for aesthetic reasons, as well as X-terminals with small screens.

iconPlacementMargin (class **IconPlacementMargin**)
This resource sets the distance between the edge of the screen and the icons that are placed along the edge of the screen. The value should be greater than or equal to 0. A default value (see below) is used if the value specified is invalid. The default value for this resource is equal to the space between icons as they are placed on the screen (this space is based on maximizing the number of icons in each row and column).

interactivePlacement (class **InteractivePlacement**)
This resource controls the initial placement of new windows on the screen. If the value is True, the pointer shape changes before a new window is placed on the screen to indicate to the user that a position should be selected for the upper-left hand corner of the window. If the value is False, windows are placed according to the initial window configuration attributes. The default value of this resource is False.

keyBindings (class **KeyBindings**)
This resource identifies the set of key bindings for window management functions. If specified, these key bindings *replace* the built-in default bindings. The named set of key bindings is specified in *mwm* resource description file. The default value for this resource is "DefaultKeyBindings".

keyboardFocusPolicy (class **KeyboardFocusPolicy**)
If set to pointer, the keyboard focus policy is to have the keyboard focus set to the client window that contains the pointer (the pointer could also be in the client window decoration that *mwm* adds). If set to explicit, the policy is to have the keyboard focus set to a client window when the user presses button 1 with the pointer on the client window or any part of the associated *mwm* decoration. The default value for this resource is explicit.

limitResize (class **LimitResize**)
If this resource is True, the user is not allowed to resize a window to greater than the maximum size. The default value for this resource is True.

lowerOnIconify (class **LowerOnIconify**)
If this resource is given the default value of True, a window's icon appears on the bottom of the window stack when the window is minimized (iconified). A value of False places the icon in the stacking order at the same place as its associated window. The default value of this resource is True.

maximumMaximumSize (class **MaximumMaximumSize**)
This resource is used to limit the maximum size of a client window as set by the user or client. The resource value is *width***x***height* (for example, 1024x1024) where the width and height are in pixels. The default value of this resource is twice the screen width and height.

moveOpaque (class **MoveOpaque**)
This resource controls whether the actual window is moved or a rectangular outline of the window is moved. A default value of False displays a rectangular outline on moves.

moveThreshold (class **MoveThreshold**)
This resource is used to control the sensitivity of dragging operations that move windows and icons. The value of this resource is the number of pixels that the locator is moved with a button down before the move operation is initiated. This is used to prevent window / icon movement when you click or double-click and there is unintentional pointer movement with the button down. The default value of this resource is 4 (pixels).

multiScreen (class **MultiScreen**)
This resource, if True, causes *mwm* to manage all the screens on the display. If False, *mwm* manages only a single screen. The default value is False.

passButtons (class **PassButtons**)
This resource indicates whether or not button press events are passed to clients after they are used to do a window manager function in the client context. If the resource value is False, the button press is not passed to the client. If the value is True, the button press is passed to the client window. The window manager function is done in either case. The default value for this resource is False.

passSelectButton (class **PassSelectButton**)
This resource indicates whether or not to pass the select button press events to clients after they are used to do a window manager function in the client context. If the resource value is False, then the button press will not be passed to the client. If the value is True, the button

press is passed to the client window. The window manager function is done in either case. The default value for this resource is True.

positionIsFrame (class **PositionIsFrame**)
This resource indicates how client window position information (from the WM_NORMAL_HINTS property and from configuration requests) is to be interpreted. If the resource value is True, the information is interpreted as the position of the MWM client window frame. If the value is False, it is interpreted as being the position of the client area of the window. The default value of this resource is True.

positionOnScreen (class **PositionOnScreen**)
This resource is used to indicate that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen (if the resource value is True). If a window is larger than the size of the screen, at least the upper-left corner of the window is on-screen. If the resource value is False, windows are placed in the requested position even if totally off-screen. The default value of this resource is True.

quitTimeout (class **QuitTimeout**)
This resource specifies the amount of time (in milliseconds) that **mwm** will wait for a client to update the WM_COMMAND property after *mwm* has sent the WM_SAVE_YOURSELF message. The default value of this resource is 1000 (ms). (Refer to the f.kill function description for additional information.)

raiseKeyFocus (class **RaiseKeyFocus**)
This resource is available only when the keyboard input focus policy is explicit. When set to True, this resource specifies that a window raised by means of the f.normalize_and_raise function also receives the input focus. The default value of this resource is False.

resizeBorderWidth (class **ResizeBorderWidth**)
This resource specifies the width (in pixels) of a client window frame border with resize handles. The specified border width includes the 3-D shadows. The default value is based on the size and resolution of the screen.

resizeCursors (class **ResizeCursors**)
This is used to indicate whether the resize cursors are always displayed when the pointer is in the window size border. If True, the cursors are shown, otherwise the window manager cursor is shown. The default value is True.

screens (class **Screens**)
This resource specifies the resource names to use for the screens managed by *mwm*. If *mwm* is managing a single screen, only the first name in the list is used. If *mwm* is managing multiple screens, the names are assigned to the screens in order, starting with screen 0. Screen 0 gets the first name, screen 1 the second name, and so on. The default screen names are 0, 1, and so on.

showFeedback (class **ShowFeedback**)
This resource controls whether or not feedback windows or confirmation dialogs are displayed. A feedback window shows a client window's initial placement and shows position and size during move and resize operations. Confirmation dialogs can be displayed

for certain operations.

The value for this resource is a list of names of the feedback options to be enabled or disabled; the names must be separated by a space. If an option is preceded by a minus sign, that option is excluded from the list. The *sign* of the first item in the list determines the initial set of options. If the sign of the first option is minus, *mwm* assumes all options are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), *mwm* starts with no options and builds up a list from the resource.

The names of the feedback options are shown below:

Name	**Description**
all	Show all feedback (Default value)
behavior	Confirm behavior switch
kill	Confirm on receipt of KILL signal
move	Show position during move
none	Show no feedback
placement	Show position and size during initial placement
quit	Confirm quitting **mwm**
resize	Show size during resize
restart	Confirm *mwm* restart

The following command line illustrates the syntax for showFeedback:

Mwm*showFeedback: placement resize behavior restart

This resource specification provides feedback for initial client placement and resize, and enables the dialog boxes to confirm the restart and set behavior functions. It disables feedback for the move function. The default value for this resource is all.

startupKeyFocus (class **StartupKeyFocus**)
This resource is available only when the keyboard input focus policy is explicit. When given the default value of True, a window gets the keyboard input focus when the window is mapped (that is, initially managed by the window manager). It is recommended that both **autoKeyFocus** and **startupKeyFocus** be True to work with tear off menus. The default value is True.

transientDecoration (class **TransientDecoration**)
This controls the amount of decoration that *mwm* puts on transient windows. The decoration specification is exactly the same as for the **clientDecoration** (client specific) resource. Transient windows are identified by the WM_TRANSIENT_FOR property, which is added by the client to indicate a relatively temporary window. The default value for this resource is menu title (that is, transient windows have frame borders and a titlebar with a window menu button).

An application can also specify which decorations *mwm* should apply to its windows. If it does so, *mwm* applies only those decorations indicated by both the application and the **transientDecoration**

resource. Otherwise, *mwm* applies the decorations indicated by the **transientDecoration** resource. For more information see the description of **XmNmwmDecorations** on the **VendorShell(3X)** reference page.

transientFunctions (class **TransientFunctions**)
This resource is used to indicate which window management functions are applicable (or not applicable) to transient windows. The function specification is exactly the same as for the **clientFunctions** (client specific) resource. The default value for this resource is -minimize -maximize.

An application can also specify which functions *mwm* should apply to its windows. If it does so, *mwm* applies only those functions indicated by both the application and the **transientFunctions** resource. Otherwise, *mwm* applies the functions indicated by the **transientFunctions** resource. For more information see the description of **XmNmwmFunctions** on the **VendorShell(3X)** reference page.

useIconBox (class **UseIconBox**)
If this resource is given a value of True, icons are placed in an icon box. When an icon box is not used, the icons are placed on the root window (default value).

wMenuButtonClick (class **WMenuButtonClick**)
This resource indicates whether a click of the mouse when the pointer is over the window menu button posts and leaves posted the window menu. If the value given this resource is True, the menu remains posted. True is the default value for this resource.

wMenuButtonClick2 (class **WMenuButtonClick2**)
When this resource is given the default value of True, a double-click action on the window menu button does an f.kill function.

Client Specific Resources

The syntax for specifying client specific resources is

`Mwm*`*`client_name_or_class`*`*`*`resource_id`

For example, **Mwm*mterm*windowMenu** is used to specify the window menu to be used with mterm clients. The syntax for specifying client specific resources for all classes of clients is

`Mwm*`*`resource_id`*

Specific client specifications take precedence over the specifications for all clients. For example, **Mwm*windowMenu** is used to specify the window menu to be used for all classes of clients that don't have a window menu specified.

The syntax for specifying resource values for windows that have an unknown name and class (that is, windows that do not have a WM_CLASS property associated with them) is

`Mwm*defaults*`*`resource_id`*

For example, **Mwm*defaults*iconImage** is used to specify the icon image to be used for windows that have an unknown name and class.

The following client specific resources can be specified:

Client Specific Resources

Name	Class	Value Type	Default
clientDecoration	ClientDecoration	string	all
clientFunctions	ClientFunctions	string	all
focusAutoRaise	FocusAutoRaise	T/F	varies
iconImage	IconImage	pathname	(image)
iconImageBackground	Background	color	icon background
iconImageBottomShadowColor	Foreground	color	icon bottom shadow
iconImageBottomShadowPixmap	BottomShadow-Pixmap	color	icon bottom shadow pixmap
iconImageForeground	Foreground	color	varies
iconImageTopShadowColor	Background	color	icon top shadow color
iconImageTopShadowPixmap	TopShadow-Pixmap	color	icon top shadow pixmap
matteBackground	Background	color	background
matteBottomShadowColor	Foreground	color	bottom shadow color
matteBottomShadowPixmap	BottomShadow-Pixmap	color	bottom shadow pixmap
matteForeground	Foreground	color	foreground
matteTopShadowColor	Background	color	top shadow color
matteTopShadowPixmap	TopShadow-Pixmap	color	top shadow pixmap
matteWidth	MatteWidth	pixels	0
maximumClientSize	MaximumClientSize	wxh vertical horizontal	fill the screen
useClientIcon	UseClientIcon	T/F	F
usePPosition	UsePPosition	string	nonzero
windowMenu	WindowMenu	string	"Default- Window-Menu"

clientDecoration (class **ClientDecoration**)

This resource controls the amount of window frame decoration. The resource is specified as a list of decorations to specify their inclusion in the frame. If a decoration is preceded by a minus sign, that decoration is excluded from the frame. The *sign* of the first item in the list determines the initial amount of decoration. If the sign of the first decoration is minus, *mwm* assumes all decorations are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), then *mwm* starts with no decoration and builds up a list from the resource.

An application can also specify which decorations *mwm* should apply to its windows. If it does so, *mwm* applies only those decorations indicated by both the application and the **clientDecoration** resource. Otherwise, *mwm* applies the decorations indicated by the **clientDecoration** resource. For more information see the description

of **XmNmwmDecorations** on the **VendorShell(3X)** reference page.

Name	Description
all	Include all decorations (default value)
border	Window border
maximize	Maximize button (includes title bar)
minimize	Minimize button (includes title bar)
none	No decorations
resizeh	Border resize handles (includes border)
menu	Window menu button (includes title bar)
title	Title bar (includes border)

Examples:

```
Mwm*XClock.clientDecoration: -resizeh -maximize
```

This removes the resize handles and maximize button from XClock windows.

```
Mwm*XClock.clientDecoration: menu minimize border
```

This does the same thing as above. Note that either **menu** or **minimize** implies **title**.

clientFunctions (class **ClientFunctions**)

This resource is used to indicate which *mwm* functions are applicable (or not applicable) to the client window. The value for the resource is a list of functions. If the first function in the list has a minus sign in front of it, then *mwm* starts with all functions and subtracts from that set. If the first function in the list has a plus sign in front of it, then *mwm* starts with no functions and builds up a list. Each function in the list must be preceded by the appropriate plus or minus sign and separated from the next function by a space.

An application can also specify which functions *mwm* should apply to its windows. If it does so, *mwm* applies only those functions indicated by both the application and the **clientFunctions** resource. Otherwise, *mwm* applies the functions indicated by the **clientFunctions** resource. For more information see the description of **XmNmwmFunctions** on the *VendorShell*( ) reference page.

The table below lists the functions available for this resource:

Name	Description
all	Include all functions (default value)
none	No functions
resize	f.resize
move	f.move
minimize	f.minimize
maximize	f.maximize
close	f.kill

focusAutoRaise (class **FocusAutoRaise**)
When the value of this resource is True, clients are raised when they get the keyboard input focus. If the value is False, the stacking of windows on the display is not changed when a window gets the keyboard input focus. The default value is True when the keyboardFocusPolicy is explicit and False when the keyboardFocusPolicy is pointer.

iconImage (class **IconImage**)
This resource can be used to specify an icon image for a client (for example, "Mwm*myclock*iconImage"). The resource value is a pathname for a bitmap file. The value of the (client specific) useClientIcon resource is used to determine whether or not user supplied icon images are used instead of client supplied icon images. The default value is to display a built-in window manager icon image.

iconImageBackground (class **Background**)
This resource specifies the background color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon background color (that is, specified by "Mwm*background or Mwm*icon*background).

iconImageBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow color (that is, specified by Mwm*icon*bottomShadowColor).

iconImageBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow Pixmap (that is, specified by Mwm*icon*bottomShadowPixmap).

iconImageForeground (class **Foreground**)
This resource specifies the foreground color of the icon image that is displayed in the image part of an icon. The default value of this

resource varies depending on the icon background.

iconImageTopShadowColor (class **Background**)
This resource specifies the top shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow color (that is, specified by Mwm*icon*topShadowColor).

iconImageTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow pixmap (that is, specified by Mwm*icon*topShadowPixmap).

matteBackground (class **Background**)
This resource specifies the background color of the matte, when **matteWidth** is positive. The default value of this resource is the client background color (that is, specified by "Mwm*background or Mwm*client*background).

matteBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow color (that is, specified by "Mwm*bottomShadowColor or Mwm*client*bottomShadowColor).

matteBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow pixmap (that is, specified by

```
"Mwm*bottomShadowPixmap or Mwm*client*bottomShadowPixmap).
```

matteForeground (class **Foreground**)
This resource specifies the foreground color of the matte, when **matteWidth** is positive. The default value of this resource is the client foreground color (that is, specified by "Mwm*foreground or Mwm*client*foreground).

matteTopShadowColor (class **Background**)
This resource specifies the top shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow color (that is, specified by "Mwm*topShadowColor or Mwm*client*topShadowColor).

matteTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow pixmap of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow pixmap (that is, specified by "Mwm*topShadowPixmap or Mwm*client*topShadowPixmap).

matteWidth (class **MatteWidth**)
This resource specifies the width of the optional matte. The default value is 0, which effectively disables the matte.

maximumClientSize (class **MaximumClientSize**)
This resource is either a size specification or a direction that indicates how a client window is to be maximized. The resource value can be specified as a size specification *width***x***height*. The width and height are interpreted in the units that the client uses (for example, for terminal emulators this is generally characters). Alternately, "vertical"

or "horizontal" can be specified to indicate the direction in which the client maximizes.

If this resource is not specified, the maximum size from the WM_NORMAL_HINTS property is used if set. Otherwise the default value is the size where the client window with window management borders fills the screen. When the maximum client size is not determined by the maximumClientSize resource, the maximumMaximumSize resource value is used as a constraint on the maximum size.

useClientIcon (class **UseClientIcon**)
If the value given for this resource is True, a client-supplied icon image takes precedence over a user-supplied icon image. The default value is False, giving the user-supplied icon image higher precedence than the client-supplied icon image.

usePPosition (class **UsePPosition**)
This resource specifies whether Mwm honors program specified position **PPosition** specified in the WM_NORMAL_HINTS property in the absence of an user specified position. Setting this resource to on, causes **mwm** to always honor program specified position. Setting this resource to off, causes *mwm* to always ignore program specified position. Setting this resource to the default value of nonzero cause *mwm* to honor program specified position other than (0,0).

windowMenu (class **WindowMenu**)
This resource indicates the name of the menu pane that is posted when the window menu is popped up (usually by pressing button 1 on the window menu button on the client window frame). Menu panes are specified in the MWM resource description file. Window menus can be customized on a client class basis by specifying resources of the form **Mwm****client_name_or_class****windowMenu** (see "Mwm Resource Description File Syntax"). The default value of this resource is "DefaultWindowMenu".

Resource Description File

The MWM resource description file is a supplementary resource file that contains resource descriptions that are referred to by entries in the defaults files (.Xdefaults, app-defaults/Mwm). It contains descriptions of resources that are to be used by *mwm*, and that cannot be easily encoded in the defaults files (a bitmap file is an analogous type of resource description file). A particular **mwm resource descriptionfile** can be selected using the **configFile** resource.

The following types of resources can be described in the **mwm** resource description file:

Buttons Window manager functions can be bound (associated) with button events.

Keys Window manager functions can be bound (associated) with key press events.

Menus Menu panes can be used for the window menu and other menus posted with key bindings and button bindings.

mwm Resource Description File Syntax

The *mwm* resource description file is a standard text file that contains items of information separated by blanks, tabs, and newline characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (for example, the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes ("). Single characters can be quoted by preceding them by the backslash character (\). All text from an unquoted **#** to the end of the line is regarded as a comment and is not interpreted as part of a resource description. If **!** is the first character in a line, the line is regarded as a comment. If a line ends in a backslash character (\), the next line is considered a continuation of that line. Window manager functions can be accessed with button and key bindings, and with window manager menus. Functions are indicated as part of the specifications for button and key binding sets, and menu panes. The function specification has the following syntax:

```
function =        function_name [function_args]
function_name =   window manager function
function_args =   {quoted_item | unquoted_item}
```

The following functions are supported. If a function is specified that isn't one of the supported functions, then it is interpreted by *mwm* as **f.nop**.

f.beep This function causes a beep.

f.circle_down [**icon** | **window**]
This function causes the window or icon that is on the top of the window stack to be put on the bottom of the window stack (so that it no longer obscures any other window or icon). This function affects only those windows and icons that obscure other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window. Secondary windows always stay on top of the associated primary window and there can be no other primary windows between the secondary windows and their primary window. If an **icon** function argument is specified, the function applies only to icons. If a **window** function argument is specified, the function applies only to windows.

f.circle_up [**icon** | **window**]
This function raises the window or icon on the bottom of the window stack (so that it is not obscured by any other windows). This function affects only those windows and icons that obscure other windows and icons, or that are obscured by other windows and icons. Secondary

windows (that is, transient windows) are restacked with their associated primary window. If an *icon* function argument is specified, the function applies only to icons. If a *window* function argument is specified, the function applies only to windows.

f.exec or **!**
This function causes *command* to be executed (using the value of the MWMSHELL environment variable if it is set, otherwise the value of the SHELL environment variable if it is set, otherwise */bin/sh*). The **!** notation can be used in place of the **f.exec** function name.

f.focus_color
This function sets the colormap focus to a client window. If this function is done in a root context, the default colormap (set up by the *X Window System* for the screen where MWM is running) is installed and there is no specific client window colormap focus. This function is treated as *f.nop* if colormapFocusPolicy is not explicit.

f.focus_key
This function sets the keyboard input focus to a client window or icon. This function is treated as *f.nop* if keyboardFocusPolicy is not explicit or the function is executed in a root context.

f.kill This function is used to terminate a client. If the WM_DELETE_WINDOW protocol is set up, the client is sent a client message event, indicating that the client window should be deleted. If the WM_SAVE_YOURSELF protocol is set up, the client is sent a client message event, indicating that the client needs to prepare to be terminated. If the client does not have the WM_DELETE_WINDOW or WM_SAVE_YOURSELF protocol set up, this function causes a client's X connection to be terminated (usually resulting in termination of the client). Refer to the description of the quitTimeout resource and the WM_PROTOCOLS property.

f.lower [**–***client* | **within** | **freeFamily**]
This function lowers a primary window to the bottom of the global window stack (where it obscures no other window) and lowers the secondary window (transient window or dialog box) within the client family. The arguments to this function are mutually exclusive.

The *client* argument indicates the name or class of a client to lower. If the *client* argument is not specified, the context that the function was invoked in indicates the window or icon to lower.

Specifying **within** lowers the secondary window within the family (staying above the parent) but does not lower the client family in the global window stack.

Specifying **freeFamily** lowers the window to the bottom of the global windows stack from its local family stack.

f.maximize
This function causes a client window to be displayed with its maximum size.

f.menu This function associates a cascading (pull-right) menu with a menu pane entry or a menu with a button or key binding. The *menu_name* function argument identifies the menu to be used.

f.minimize
This function causes a client window to be minimized (iconified).

When a window is minimized when no icon box is used, its icon is placed on the bottom of the window stack (so that it obscures no other window). If an icon box is used, the client's icon changes to its iconified form inside the icon box. Secondary windows (that is, transient windows) are minimized with their associated primary window. There is only one icon for a primary window and all its secondary windows.

f.move This function causes a client window to be interactively moved.

f.next_cmap

This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key [**icon** | **window** | **transient**]

This function sets the keyboard input focus to the next window / icon in the set of windows / icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as *f.nop* if keyboardFocusPolicy is not explicit. The keyboard input focus is moved only to windows that do not have an associated secondary window that is application modal. If the **transient** argument is specified, transient (secondary) windows are traversed (otherwise, if only **window** is specified, traversal is done only to the last focused window in a transient group). If an **icon** function argument is specified, the function applies only to icons. If a **window** function argument is specified, the function applies only to windows.

f.nop This function does nothing.

f.normalize

This function causes a client window to be displayed with its normal size. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.normalize_and_raise

This function causes the corresponding client window to be displayed with its normal size and raised to the top of the window stack. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.pack_icons

This function is used to relayout icons (based on the layout policy being used) on the root window or in the icon box. In general this causes icons to be "packed" into the icon grid.

f.pass_keys

This function is used to enable / disable (toggle) processing of key bindings for window manager functions. When it disables key binding processing, all keys are passed on to the window with the keyboard input focus and no window manager functions are invoked. If the *f.pass_keys* function is invoked with a key binding to disable key-binding processing, the same key binding can be used to enable key-binding processing.

f.post_wmenu

This function is used to post the window menu. If a key is used to post the window menu and a window menu button is present, the window menu is automatically placed with its top-left corner at the bottom-left corner of the window menu button for the client window.

If no window menu button is present, the window menu is placed at the top-left corner of the client window.

f.prev_cmap

This function installs the previous colormap in the list of colormaps for the window with the colormap focus.

f.prev_key [**icon** | **window** | **transient**]

This function sets the keyboard input focus to the previous window / icon in the set of windows / icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as *f.nop* if keyboardFocusPolicy is not explicit. The keyboard input focus is moved only to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, transient (secondary) windows are traversed (otherwise, if only *window* is specified, traversal is done only to the last focused window in a transient group). If an *icon* function argument is specified, the function applies only to icons. If an *window* function argument is specified, the function applies only to windows.

f.quit_mwm

This function terminates *mwm* (but NOT the X window system).

f.raise [**–***client* | **within** | **freeFamily**]

This function raises a primary window to the top of the global window stack (where it is obscured by no other window) and raises the secondary window (transient window or dialog box) within the client family. The arguments to this function are mutually exclusive.

The *client* argument indicates the name or class of a client to lower. If the *client* is not specified, the context that the function was invoked in indicates the window or icon to lower.

Specifying **within** raises the secondary window within the family but does not raise the client family in the global window stack.

Specifying **freeFamily** raises the window to the top of its local family stack and raises the family to the top of the global window stack.

f.raise_lower [**within** | **freeFamily**]

This function raises a primary window to the top of the global window stack if it is partially obscured by another window; otherwise, it lowers the window to the bottom of the window stack. The arguments to this function are mutually exclusive.

Specifying **within** raises a secondary window within the family (staying above the parent window), if it is partially obscured by another window in the application's family; otherwise, it lowers the window to the bottom of the family stack. It has no effect on the global window stacking order.

Specifying **freeFamily** raises the window to the top of its local family stack, if obscured by another window, and raises the family to the top of the global window stack; otherwise, it lowers the window to the bottom of its local family stack and lowers the family to the bottom of the global window stack.

f.refresh

This function causes all windows to be redrawn.

f.refresh_win
This function causes a client window to be redrawn.

f.resize This function causes a client window to be interactively resized.

f.restore
This function restores the previous state of an icon's associated window. If a maximized window is iconified, then **f.restore** restores it to its maximized state. If a normal window is iconified, then **f.restore** restores it to its normalized state.

f.restore_and_raise
This function restores the previous state of an icon's associated window and raises the window to the top of the window stack. If a maximized window is iconified, then **f.restore_and_raise** restores it to its maximized state and raises it to the top of the window stack. If a normal window is iconified, then **f.restore_and_raise** restores it to its normalized state and raises it to the top of the window stack.

f.restart This function causes *mwm* to be restarted (effectively terminated and re-executed).

f.screen [**next** | **prev** | **back** *screen_number*]
This function causes the pointer to be warp to a specific screen number or to the **next**, **previous**, or last visited (**back**) screen. The arguments to this function are mutually exclusive.

The *screen_number* argument indicates the screen number that the pointer is to be warped. Screens are numbered starting from screen 0.

Specifying **next** cause the pointer to warp to the next managed screen (skipping over any unmanaged screens).

Specifying **prev** cause the pointer to warp to the previous managed screen (skipping over any unmanaged screens).

Specifying **back** cause the pointer to warp to the last visited screen.

f.send_msg *message_number*
This function sends a client message of the type _MOTIF_WM_MESSAGES with the *message_type* indicated by the *message_number* function argument. The client message is sent only if *message_number* is included in the client's _MOTIF_WM_MESSAGES property. A menu item label is grayed out if the menu item is used to do *f.send_msg* of a message that is not included in the client's _MOTIF_WM_MESSAGES property.

f.separator
This function causes a menu separator to be put in the menu pane at the specified location (the label is ignored).

f.set_behavior
This function causes the window manager to restart with the default behavior (if a custom behavior is configured) or revert to the custom behavior. By default this is bound to **Shift Ctrl Meta <Key>!**.

f.title This function inserts a title in the menu pane at the specified location.

Each function may be constrained as to which resource types can specify the function (for example, menu pane) and also what context the function can be used in (for example, the function is done to the selected client window). Function contexts are

root	No client window or icon has been selected as an object for the function.
window	A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are applied only when the window is in its normalized state (for example, *f.maximize*) or its maximized state (for example, *f.normalize*).
icon	An icon has been selected as an object for the function.

If a function's context has been specified as **icon|window** and the function is invoked in an icon box, the function applies to the icon box, not to the icons inside.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply, the function is treated as *f.nop*. The following table indicates the resource types and function contexts in which window manager functions apply.

Function	**Contexts**	**Resources**
f.beep	root, icon, window	button, key, menu
f.circle_down	root, icon, window	button, key, menu
f.circle_up	root, icon, window	button, key, menu
f.exec	root, icon, window	button, key, menu
f.focus_color	root, icon, window	button, key, menu
f.focus_key	root, icon, window	button, key, menu
f.kill	icon, window	button, key, menu
f.lower	icon, window	button, key, menu
f.maximize	icon, window(normal)	button, key, menu
f.menu	root, icon, window	button, key, menu
f.minimize	window	button, key, menu
f.move	icon, window	button, key, menu
f.next_cmap	root, icon, window	button, key, menu
f.next_key	root, icon, window	button, key, menu
f.nop	root, icon, window	button, key, menu
f.normalize	icon, window(maximized)	button, key, menu
f.normalize_and_raise	icon, window	button, key, menu
f.pack_icons	root, icon, window	button, key, menu
f.pass_keys	root, icon, window	button, key, menu
f.post_wmenu	root, icon, window	button, key
f.prev_cmap	root, icon, window	button, key, menu
f.prev_key	root, icon, window	button, key, menu
f.quit_mwm	root, icon, window	button, key, menu (root only)
f.raise	icon, window	button, key, menu
f.raise_lower	icon, window	button, key, menu
f.refresh	root, icon, window	button, key, menu
f.refresh_win	window	button, key, menu
f.resize	window	button, key, menu
f.restore	icon, window	button, key, menu

f.restore_and_raise	icon, window	button, key, menu
f.restart	root, icon, window	button, key, menu (root only)
f.screen	root, icon, window	button, key, menu
f.send_msg	icon, window	button, key, menu
f.separator	root, icon, window	menu
f.set_behavior	root, icon, window	button, key, menu
f.title	root, icon, window	menu

Window Manager Event Specification

Events are indicated as part of the specifications for button and key-binding sets, and menu panes.

Button events have the following syntax:

```
button =          [modifier_list]<button_event_name>
modifier_list =   modifier_name{modifier_name}
```

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the button event occurs). The following table indicates the values that can be used for *modifier_name*.[1] The [Alt] key is frequently labeled [Extend] or [Meta]. Alt and Meta can be used interchangeably in event specification.

Modifier	**Description**
Ctrl	Control Key
Shift	Shift Key
Alt	Alt / Meta Key
Meta	Meta / Alt Key
Lock	Lock Key
Mod1	Modifier1
Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

The following table indicates the values that can be used for *button_event_name.*

Button	Description
Btn1Down	Button 1 Press
Btn1Up	Button 1 Release
Btn1Click	Button 1 Press and Release
Btn1Click2	Button 1 Double-Click
Btn2Down	Button 2 Press
Btn2Up	Button 2 Release
Btn2Click	Button 2 Press and Release
Btn2Click2	Button 2 Double-Click
Btn3Down	Button 3 Press
Btn3Up	Button 3 Release
Btn3Click	Button 3 Press and Release
Btn3Click2	Button 3 Double-Click
Btn4Down	Button 4 Press
Btn4Up	Button 4 Release
Btn4Click	Button 4 Press and Release
Btn4Click2	Button 4 Double-Click
Btn5Down	Button 5 Press
Btn5Up	Button 5 Release
Btn5Click	Button 5 Press and Release
Btn5Click2	Button 5 Double-Click

Key events that are used by the window manager for menu mnemonics and for binding to window manager functions are single key presses; key releases are ignored. Key events have the following syntax:

key = **[***modifier_list***]**<Key>*key_name*
modifier_list = *modifier_name* **{***modifier_name***}**

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). Modifiers for keys are the same as those that apply to buttons. The *key_name* is an X11 keysym name. Keysym names can be found in the **<keysymdef.h>** file (remove the *XK_* prefix).

Button Bindings

The **buttonBindings** resource value is the name of a set of button bindings that are used to configure window manager behavior. A window manager function can be done when a button press occurs with the pointer over a framed client window, an icon, or the root window. The context for indicating where the button press applies is also the context for invoking the window manager function when the button press is done (significant for functions that are context sensitive).

The button binding syntax is

Buttons *bindings_set_name*
{
button *context* *function*
button *context* *function*
.
.

```
    button      context      function
}
```

The syntax for the *context* specification is

```
context =   object[ | context]
object =    root | icon | window | title | frame | border | app
```

The context specification indicates where the pointer must be for the button binding to be effective. For example, a context of **window** indicates that the pointer must be over a client window or window management frame for the button binding to be effective. The **frame** context is for the window management frame around a client window (including the border and titlebar), the **border** context is for the border part of the window management frame (not including the titlebar), the **title** context is for the title area of the window management frame, and the **app** context is for the application window (not including the window management frame).

If an *f.nop* function is specified for a button binding, the button binding is not done.

Key Bindings

The **keyBindings** resource value is the name of a set of key bindings that are used to configure window manager behavior. A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid contexts are the same as those that apply to button bindings.

The key binding syntax is

```
Keys  bindings_set_name
{
    key     context     function
    key     context     function
                .
                .
    key     context     function
}
```

If an *f.nop* function is specified for a key binding, the key binding is not done. If an *f.post_wmenu* or *f.menu* function is bound to a key, *mwm* will automatically use the same key for removing the menu from the screen after it has been popped up.

The *context* specification syntax is the same as for button bindings. For key bindings, the **frame**, **title**, **border**, and **app** contexts are equivalent to the **window** context. The context for a key event is the window or icon that has the

keyboard input focus (**root** if no window or icon has the keyboard input focus).

Menu Panes

Menus can be popped up using the *f.post_wmenu* and *f.menu* window manager functions. The context for window manager functions that are done from a menu is *root*, *icon* or *window* depending on how the menu was popped up. In the case of the *window* menu or menus popped up with a key binding, the location of the keyboard input focus indicates the context. For menus popped up using a button binding, the context of the button binding is the context of the menu.

The menu pane specification syntax is

Menu *menu_name*
{
label [*mnemonic*] [*accelerator*] *function*
label [*mnemonic*] [*accelerator*] *function*
.
.
label [*mnemonic*] [*accelerator*] *function*
}

Each line in the *Menu* specification identifies the label for a menu item and the function to be done if the menu item is selected. Optionally a menu button mnemonic and a menu button keyboard accelerator may be specified. Mnemonics are functional only when the menu is posted and keyboard traversal applies.

The *label* may be a string or a bitmap file. The label specification has the following syntax:

```
label =         text | bitmap_file
bitmap_file =   @file_name
text =          quoted_item | unquoted_item
```

The string encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the *f.nop* function or an invalid function or a function that doesn't apply in the current context.

A *mnemonic* specification has the following syntax

```
mnemonic =   _character
```

The first matching *character* in the label is underlined. If there is no matching *character* in the label, no mnemonic is registered with the window manager for that label. Although the *character* must exactly match a character in the label, the mnemonic does not execute if any modifier (such as Shift) is pressed with the character key.

The *accelerator* specification is a key event specification with the same syntax as is used for key bindings to window manager functions.

Environment

mwm uses the environment variable HOME specifying the user's home directory.

mwm uses the environment variable LANG specifying the user's choice of language for the *mwm* message catalog and the *mwm* resource description file.

mwm uses the environment variables XFILESEARCHPATH, XUSERFILESEARCHPATH, XAPPLRESDIR, XENVIRONMENT, LANG, and HOME in determining search paths for resource defaults files. *mwm* may also use XBMLANGPATH to search for bitmap files.

mwm reads the $HOME/.motifbind file if it exists to install a virtual key bindings property on the root window. For more information on the content of the .motifbind file, see **VirtualBindings(3X)**.

mwm uses the environment variable MWMSHELL (or SHELL, if MWMSHELL is not set), specifying the shell to use when executing commands via the *f.exec* function.

Files

/usr/lib/X11/$LANG/system.mwmrc
/usr/lib/X11/system.mwmrc
/usr/lib/X11/app-defaults/Mwm
$HOME/Mwm
$HOME/.Xdefaults
$HOME/$LANG/.mwmrc
$HOME/.mwmrc
$HOME/.motifbind

SEE ALSO

X

VendorShell(3X), *VirtualBindings*(3X), *XmInstallImage*(3X).

NAME

newgrp – change to a new group

SYNOPSIS

`newgrp [-l] group ...`

DESCRIPTION

The *newgrp* utility changes the shell execution environment to have the specified real and effective *group*. This new shell environment keeps the working directory, file creation mask, and exported variables of the previous environment. Even if the *newgrp* fails to assign a new group, it still creates a new shell environment.

newgrp takes the following options:

–l Change back to the login environment; that is, the environment the user gets when logging in.

The *group* is the name or numeric group ID of a group in the group database. If you don't specify a *group*, *newgrp* changes the real and effective group ID back to your login group.

If you are not listed as a member of the group and the group requires a password, then you are prompted to enter the password. If you're listed as a member of the group, you're not prompted.

DIAGNOSTICS

If it succeeds, the exit status is the exit status of the shell. Otherwise, it returns a value >0.

NOTES

Because of the security mechanisms of Windows NT, there is no way to get the appropriate permissions to change groups. Therefore, this command always fails to change the effective group. It does create a new shell environment, as described above.

SEE ALSO

sh, *exec*

NAME

nice – run a command with a non-standard priority

SYNOPSIS

```
nice [–n increment] utility [argument ...]
nice [–increment] utility [argument ...]
```

DESCRIPTION

The *nice* command runs the specified *utility* with the given *argument*s using a different scheduling priority. The *increment* is used to specify the change in priority. It can only be run successfully by a user with the appropriate privileges.

To give a command a higher priority use a negative *increment*.

If you don't specify an *increment*, the command runs at a slightly reduced priority.

To change the priority of a running process, use the *renice* command.

Nice takes the following options:

increment

–n *increment* Alter the utility's status by *increment*. A negative increment gives the command a higher priority, so it runs at the expense of other processes; a positive increment gives the command a lower priority, so other processes receive precedence of *utility*.

DIAGNOSTICS

If *nice* succeeds in running *utility*, it returns the exit status of *utility*. Otherwise, it returns one of these values:

1–125 An error occurred in *nice* itself.

126 The *utility* could not be invoked.

127 The *utility* could not be found.

NOTE

Currently, the INTERIX subsystem only supports one priority level and there are no privileges defined that can alter that priority level.

SEE ALSO

renice

NAME

nl – filter to number lines

SYNOPSIS

```
nl [-p] [-b type] [-d delim] [-h type]
   [-i incr] [-l num] [-n  format] [-s sep]
   [-v startnum] [-w width] [file]
```

DESCRIPTION

The *nl* utility reads lines from standard input or from *file* and writes those lines to standard output, with line numbers added on the left.

The input text is treated in terms of *logical pages*.

Line numbers restart at the beginning of each logical page; a logical page contains a header, a body, and a footer, any of which can be empty. Each section of a logical page can have its own numbering — for instance, headers may have no numbers, while blank lines are numbered in the body but not in the footer.

The sections of a logical page are indicated by the following delimiters, which occur alone on an input line:

Line	*Start of*
`\:\:\:`	header
`\:\:`	body
`\:`	footer

By default, a file contains only one logical page body.

Nl takes the following options:

–b *type* Specify the logical body lines to be numbered. Values for *type* are:

- **a** All lines are numbered.
- **n** No line numbering.
- **p***regexp* Number only the lines containing the specified regular expression *regexp*.
- **t** Number only lines consisting entirely of text characters classified as **graph** characters in the current locale.

–d *delim* Set the delimiter characters from `\:` to the one or two characters specified by *delim*. If only one character is specified, the second is a colon (:).

–f *type* Specify the logical footer to be numbered. The values of *type* are the same as for **–b**.

–h *type* Specify numbering for the logical header section. The values of *type* are the same as for **–b**.

–i *incr* Increment the numbers of the logical page lines by *incr*. The default is 1.

–l *num* Treat *num* blank lines as 1. The default is 1. A value of **–l 3** means that only the third adjacent blank line is numbered, if the logical section has numbering turned on.

–n *format*
Number the lines in the specified *format*. Values for *format* are:

ln Left justified with leading zeroes suppressed.
rn Right justified with leading zeroes suppressed. This is the default.
rz Right justified with leading zeroes retained.

–p Preserve numbering — do not restart it at logical page delimiters.

–s *sep* Separate the line number and the text characters by the characters in the string *sep*. The default value of *sep* is a tab character.

–v *startnum*
Begin numbering logical page lines with the value *startnum*. The default is 1.

–w *width* Use *width* characters for the line number. The default is 6.

DIAGNOSTICS

The *nl* utility returns 0 on success and >0 if an error occurred.

SEE ALSO

cat, *pr*

NAME

nm – list symbolic names stored in an object file

SYNOPSIS

`nm [-AefgnoPuvx] [-t format] file ...`

DESCRIPTION

Nm displays symbolic information stored in the object file, executable file, or an object-file library specified by *file*. Note that a file may not have any symbol information stored in it; a file must be created with symbolic information (for example, the **–g** option to *c89*, *cc*, and *gcc* stores symbolic information in created object files).

Nm takes the following options:

–A Display the full pathname or library name of the object on each line.

–e Display only external and static symbols.

–f Use the full output format, including symbols normally suppressed, such as .text.

–g Display only the external (global) symbols.

–o Display numeric values in octal (this is equivalent to **–t o**).

–P Use the portable format; see the description for more details.

–t *format*
Use the specified *format* for numeric values. The *format* is one of **d** (decimal — the default), **o** (octal), or **x** (hexadecimal).

–u Display only those symbols that are undefined.

–v Display the output sorted by value rather than alphabetically.

–x Display numeric values in hexadecimal (this is equivalent to **–t x**).

The output is sorted alphabetically by symbol name. For each symbol, *nm* writes:

```
name value type symbol_name
```

name The name of the file; displayed only only if the **–A** option is given.
value Offset in the the file; the default format is decimal, though this can be changed by the **–o**, **–t**, or **–x** options.
type The symbol's type, which is one of:

- **A** Absolute symbol, global.
- **a** Absolute symbol, local.
- **B** BSS (uninitialized data space) symbol, global.
- **b** BSS (uninitialized data space) symbol, local.
- **D** Data symbol, global.
- **d** Data symbol, local.
- **T** Text symbol, global.
- **t** Text symbol, local.
- **U** Undefined symbol.

symbol_name

The symbol name.

If you specify more than one *file* or if *file* is a library, then *nm* displays the file name or the library and module name before the list of symbols for that file or module.

If you specify the **–A** option, *nm* writes the file name before each line of output.

The **–P** option changes the output format to:

```
name type value
```

The default format for *value* is hexadecimal when you specify **–P**. Multiple *file*s and the **–A** flag are handled in the same as normal output.

DIAGNOSTICS

Nm returns 0 if it completed successfully, and >0 if an error occurred. An object file without symbol information is not an error.

SEE ALSO

ar, *c89*, *gcc*

NAME

nohup – invoke a utility so it is immune to hangups

SYNOPSIS

nohup *command* **[***arg ...***]**

DESCRIPTION

The *nohup* utility invokes *utility* with its arguments and at this time sets the signal SIGHUP to be ignored. The signal SIGQUIT may also be set to be ignored. If the standard output is a terminal, the standard output is appended to the file **nohup.out** in the current directory. If standard error is a terminal, it is directed to the same place as the standard output.

DIAGNOSTICS

Nohup exits 1 if an error occurs, otherwise the exit status is that of *command*.

ENVIRONMENT

Nohup uses the value of the following variable:

HOME If the output file **nohup.out** cannot be created in the current directory, the *nohup* utility creates it in the directory named by *HOME*.

SEE ALSO

kill, *sh*, *strsignal*

NAME

chgpath, ntpath2posix, posixpath2nt – convert pathnames between Win32 format and POSIX format

SYNOPSIS

`chgpath [-n|-p] [-l|–u|–c] path`

`ntpath2posix [-l|–u|–c] path`

`posixpath2nt [-l|–u] path`

DESCRIPTION

The *chgpath* utility converts a pathname between the Win32 format (for example, **c:\Interix\usr\share**) and the POSIX format (for example, **//C/Interix/usr/share**). The converted pathname is written to standard output.

If called as *ntpath2posix*, it converts from Win32 format to POSIX format. If called as *posixpath2nt*, it converts from POSIX format to Win32 format.

By default, the utility maintains the case of the *path*, although when converting to POSIX format, it will *always* produce an uppercase drive letter.

The options are:

–c Adjust case to match an existing pathname. (This only applies to **chgpath -n** and *ntpath2posix*.) After translating the pathname into POSIX format, the utility adjusts the case on a component-by-component basis. Non-existent components are not changed. The **–c** option takes precedence over **–l** and **–u**.

–l Produces a filename that is in lowercase. For *ntpath2posix*, the drive letter is *always* an uppercase letter, even if this option is given.

–n When given to *chgpath*, converts a POSIX-style pathname to a Win32-style pathname (*posixpath2nt*).

–p When given to *chgpath*, converts a Win32-style pathname to a POSIX-style pathname (*ntpath2posix*).

–u Produces a filename that is in uppercase.

These commands are normally used in shell scripts that, in turn, invoke Win32 commands.

DIAGNOSTICS

Chgpath returns 0 for success and >0 for failure.

EXAMPLES

To set the current *TMP* environment variable in a form that can be understood by Win32 programs: export TMP=$(posixpath2nt $TMP)

To return it to the form expected by the INTERIX utilities:

```
export TMP=$(ntpath2posix $TMP)
```

Assuming INTERIX is installed in **//C/Interix**:

```
$ ntpath2posix -c //C/interix
//C/Interix
```

NOTES

The **–c** option matches the first pathname component returned by *readdir*( ). It is not recursive, so sufficiently-complicated file hierarchies can fool it. Suppose you have the files and directories:

```
C:\Something
C:\Something\BIN
C:\Something\BIN\file
C:\Something\BIn     file
C:\Something\Bin
C:\Something\Bin\MyStuff
C:\Soemthing\Bin\MyStuff\myProg
```

This command will not work:

```
$ ntpath2posix -c C:\something\bin\mystuff\myprog
//C/Something/Bin/myStuff/myProg
```

This is still a useful option without providing a general replacement for *find*.

SEE ALSO

find

NAME

objdump – display information from object files.

SYNOPSIS

```
objdump [-a|--archive-headers]
        [-b bfdname|--target=bfdname]
        [-C|--demangle] [--debugging]
        [-d|--disassemble] [-D|--disassemble-all]
        [--disassemble-zeroes]
        [-EB|-EL|--endian={big|little}]
        [-f|--file-headers]
        [-h|--section-headers|--headers]
        [-i|--info] [-j section|--section=section]
        [-l|--line-numbers]
        [-m machine|--architecture=machine]
        [--prefix-addresses]
        [-r|--reloc] [-R|--dynamic-reloc]
        [-s|--full-contents]
        [-S|--source]
        [--[no-]show-raw-insn]
        [--stabs] [-t|--syms]
        [-T|--dynamic-syms]
        [-x|--all-headers]
        [--start-address=address]
        [--stop-address=address]
        [--adjust-vma=offset]
        [--version] [--help] objfile ...
```

DESCRIPTION

Objdump displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work.

Objfile... are the object files to be examined. When you specify archives, *objdump* shows information on each of the member object files.

OPTIONS

Where long and short forms of an option are shown together, they are equivalent. At least one option besides **–l** (**––line–numbers**) must be given.

–a, **––archive–headers**

If any files from *objfile* are archives, display the archive header information (in a format similar to "**ls –l**"). Besides the information you could list with "**ar tv**", *objdump* **–a** shows the object file format of each archive member.

––adjust–vma=*offset*

When dumping information, first add *offset* to all the section addresses. This is useful if the section addresses do not correspond to the symbol table, which can happen when putting sections at particular addresses when using a format which can not represent section addresses, such as a.out.

–b *bfdname*, **––target=***bfdname*
Specify the object-code format for the object files to be *bfdname*. This may not be necessary; *objdump* can automatically recognize many formats. For example,

```
objdump –b oasys –m vax –h fu.o
```

display summary information from the section headers (**–h**) of **fu.o**, which is explicitly identified (**–m**) as a Vax object file in the format produced by Oasys compilers. You can list the formats available with the **–i** option.

–C, **––demangle**
Decode (*demangle*) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable.

––debugging
Display debugging information. This attempts to parse debugging information stored in the file and print it out using a C like syntax. Only certain types of debugging information have been implemented.

–d, **––disassemble**
Display the assembler mnemonics for the machine instructions from *objfile*. This option only disassembles those sections which are expected to contain instructions.

–D, **––disassemble-all**
Like **–d**, but disassemble the contents of all sections, not just those expected to contain instructions.

––disassemble–zeroes
Normally the disassembly output will skip blocks of zeroes. This option directs the disassembler to disassemble those blocks, just like any other data.

–EB, **––endian=big**

–EL, **––endian=little**
Specify the endianness of the object files. This only affects disassembly. This can be useful when disassembling a file format which does not describe endianness information, such as S-records.

–f, **––file–headers**
Display summary information from the overall header of each file in *objfile*.

–h, **––section–headers**, ––headers
Display summary information from the section headers of the object file.

––help
Print a summary of the options to *objdump* and exit.

–i, **––info**
Display a list showing all architectures and object formats available for specification with **–b** or **–m**.

–j *name*, **––section=***name*
Display information only for section *name.*

–l, **––line–numbers**
Label the display (using debugging information) with the filename and source line numbers corresponding to the object code shown. Only useful with **–d**, **–D**, or **–r**.

–m *machine*, **––architecture=***machine*
Specify the architecture to use when disassembling object files. This can be useful when disasembling object files which do not describe architecture information, such as S-records. You can list the available architectures with the **–i** option.

––prefix–addresses
When disassembling, print the complete address on each line. This is the older disassembly format.

–r, **– –reloc**
Print the relocation entries of the file. If used with **–d** or **–d**, the relocations are printed interspersed with the disassembly.

–R, **––dynamic–reloc**
Print the dynamic relocation entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries.

–s, **––full–contents**
Display the full contents of any sections requested.

–S, **––source**
Display source code intermixed with disassembly, if possible. Implies **–d**.

––show–raw–insn
When disassembling instructions, print the instruction in hex as well as in symbolic form. This is the default except when **—prefix–addresses** is used.

––no–show–raw–insn
When disassembling instructions, do not print the instruction bytes. This is the default when **—prefix–addresses** is used.

––stabs Display the contents of the .stab, .stab.index, and .stab.excl sections from an ELF file. This is only useful on systems (such as Solaris 2.0) in which .stab debugging symbol-table entries are carried in an ELF section. In most other file formats, debugging symbol-table entries are interleaved with linkage symbols, and are visible in the —syms output.

––start–address=*address*
Start displaying data at the specified address. This affects the output of the **–d**, **–r** and **–s** options.

––stop–address=*address*
Stop displaying data at the specified address. This affects the output of the **–d**, **–r** and **–s** options.

–t, **––syms**
Symbol Table. Print the symbol table entries of the file. This is similar to the information provided by the *nm* program.

–T, **––dynamic–syms**
Dynamic Symbol Table. Print the dynamic symbol table entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries. This is similar to the information provided by the *nm* program when given the **–D** (**––dynamic**) option.

––version
Print the version number of *objdump* and exit.

–x, **––all–headers**
Display all available header information, including the symbol table and relocation entries. Using **–x** is equivalent to specifying all of **–a –f –h –r –t**.

COPYING

Copyright (c) 1991, 92, 93, 94, 95, 1996 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

SEE ALSO

nm

binutils entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991).

NAME

oclock – display time of day

SYNOPSIS

`oclock [-option ...]`

DESCRIPTION

Clock simply displays the current time on an analog display.

OPTIONS

–fg *foreground_color*
choose a different color for the both hands and the jewel of the clock

–bg *background_color*
choose a different color for the background.

–jewel *jewel_color*
choose a different_color for the jewel on the clock.

–minute *minute_color*
choose a different color for the minute hand of the clock.

–hour *hour_color*
choose a different color for the hour hand of the clock.

–backing { WhenMapped Always NotUseful }
selects an appropriate level of backing store.

–geometry *geometry*
define the initial window geometry; see *X(1)*.

–display *display*
specify the display to use; see *X*.

–bd *border*
choose a different color for the window border.

–bw *border*
choose a different width for the window border. As the Clock widget changes its border around quite a bit, this is most usefully set to zero.

–noshape
causes the clock to not reshape itself and ancestors to exactly fit the outline of the clock.

–transparent
causes the clock to consist only of the jewel, the hands, and the border.

COLORS

If you would like your clock to be viewable in color, include the following in the #ifdef COLOR section you read with *xrdb*:

```
*customization:                    -color
```

This will cause oclock to pick up the colors in the app-defaults color customization file: **$OPENNT_ROOT/usr/lib/X11/app-defaults/Clock-color**. Below are the default colors:

```
Clock*Background: grey
Clock*BorderColor: light blue
Clock*hour: yellow
Clock*jewel: yellow
Clock*minute: yellow
```

SEE ALSO

X, X Toolkit documentation

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Keith Packard, MIT X Consortium

NAME

od – octal, decimal, hex, ascii dump

SYNOPSIS

od **[**-v**]** **[**-A *addr-base***]** **[**-j *skip***]** **[**-N *count***]** **[**-t *typestring***]** *... [file ...]*

od **[**–aBbcDdeFfHhIiLlOovXx**]** **[[**+**]***offset***[**.**]**Bb**]]** *file*

DESCRIPTION

The *od* utility is mapped to the *hexdump* utility. As well as supporting the new POSIX.2 standard options, *od* also supports most of the older Berkeley options (the second synopsis line). In the older syntax, you can supply an offset from the beginning of the file. If no file name is specified, you must place a **+** before the offset. The *offset* is assumed to be an octal value, unless it is followed by a decimal point (**.**), in which case it is assumed to be decimal. If the *offset* is followed by a **b**, it is treated as a number of 512-byte blocks — for example, "10.b" represents an offset of 5120 bytes.

–A *addr-base*
: Specify the input address base (**d**, **o**, **x**).

–a Display bytes in ASCII. Control characters are translated to a three-letter abbreviation (such as "soh", "etx" and so on).

–B Display in unsigned octal words of 2 bytes.

–b Display unsigned bytes in octal.

–c Display bytes in ASCII. Control characters are translated to backslash-escape sequences or to a raw octal number.

–D Display unsigned decimal long words of 4 bytes.

–d Display unsigned decimal words of 2 bytes.

–e Display floating point words of 8 bytes. In *printf*( ) notation, they are displayed as %21.14e.

–F Display floating point words of 8 bytes. In *printf*( ) notation, they are displayed as %21.14e.

–f Display floating point words of 4 bytes. In *printf*( ) notation, they are displayed as %14.7e.

–H Display in unsigned hexadecimal long words of 4 bytes.

–h Display in unsigned hexadecimal words of 2 bytes.

–I Display in decimal long words of 4 bytes.

–j *skip* Jump over *skip* bytes from the beginning of the input.

–L Display in decimal long words of 4 bytes. (Same as **–I**).

–l Display in decimal long words of 4 bytes. (Same as **–I**).

–N *count*
Format no more than *count* bytes of data.

–O Display in unsigned octal long words of 4 bytes.

–o Display in unsigned octal words of 2 bytes.

–s Display in signed decimal words of 2 bytes.

–S Display in signed decimal long words of 4 bytes.

–t *typestring*
Specify one or more output types. Values for *typestring* may be:

a ASCII characters from the ISO 646 character set.
c Characters; non-printable characters are displayed as backslash sequences.
d[*n*]
Signed decimal; the single digit *n* is optional, and specifies how many bytes to use. The size of *n* must be the size of a **char**, **short**, **int**, or **long**. The default is the size of an **int**.
f[*n*]
Floating point; the single digit *n* is optional, and specifies how many bytes to use. The size of *n* must be the size of a **float**, **double**, or a **long double**. The default is the size of an **double**.
o[*n]*
Octal; the single digit *n* is optional, and specifies how many bytes to use. The size of *n* must be the size of a **char**, **short**, **int**, or **long**. The default is the size of an **int**.
u[*n]*
Unsigned decimal; the single digit *n* is optional, and specifies how many bytes to use. The size of *n* must be the size of a **char**, **short**, **int**, or **long**. The default is the size of an **int**.
x[*n]*
Hexadecimal; the single digit *n* is optional, and specifies how many bytes to use. The size of *n* must be the size of a **char**, **short**, **int**, or **long**. The default is the size of an **int**.

The default value is **–t o2**.

–v Write all input data. Without this option, *od* displays a single "*" for a line which is identical to the preceding line except for address.

–x Display in unsigned hexadecimal words of 2 bytes (same as **–h**).

–X Display in unsigned hexadecimal long words of 4 bytes (same as **–H**).

The *od* utility is really the hexdump*command*by invoked as *od*. It does not provide compatibility for the **–P**, **–p**, or **–w** options, nor for the ''label'' component of the offset syntax.

SEE ALSO
hexdump

NAME

openntreg – Enable INTERIX subsystem

SYNOPSIS

```
openntreg –x|–h

openntreg [–B|–S|–T] [–s] license-key
```

DESCRIPTION

The *openntreg* utility overwrites any previous license keys with the new *license-key*. The license key enables the INTERIX subsystem, and contains some information about the product.

The *openntreg* utility is most often used to convert a demonstration copy into purchased copy of the product, or to extend the demonstration period. The *license-key* is obtained from Softway Systems. For more information, see the *Getting Started* guide (**$OPENNT_ROOT/pubs/getstart.doc** or **$OPENNT_ROOT/pubs/getstart.pdf**) or the INTERIX *User's Guide* (**$OPENNT_ROOT/pubs/cuintro.pdf**).

The *openntreg* takes the following options:

–B Installs the given key for the INTERIX base product (INTERIX Workstation Lite, INTERIX Workstation, or INTERIX Server). This is the default if no product is selected.

–h Print a help explanation.

–S Installs the given key for the INTERIX Software Development Kit.

–s Silent operation; does not produce error messages.

–T Installs the given key for the INTERIX Telnet Server.

–x Print a help explanation.

DIAGNOSTICS

The *openntreg* utility exits with status 0 for success, and >0 if an error occurred.

NOTES

The *openntreg* program is a Win32 program; it will run even if the demonstration period for the INTERIX has expired.

If the *Getting Started* or *User's Guide* files were not installed on your system, they can be read directly from the INTERIX CD-ROM. The **getstart.doc** file can be read using the Windows NT WordPad accessory. The ***.pdf** files can be read using the Adobe Acrobat Reader.

NAME

pack – compress files

SYNOPSIS

`pack [-f] [-] file ...`

DESCRIPTION

The *pack* utility compresses the specified *file*s and stores the compressed versions in files identified by the name *file***.z**. The specified *file* can include the **.z** suffix. If the calling processs has the appropriate privileges, the original file is removed. To restore a packed file, use *unpack* or *pcat*.

For each packed file, *pack* writes to standard output one of the lines:

```
pack: file: percent% Compression
or
pack: file: no saving - file unchanged
```

Where *file* is the filename and *percent* is the percent compression.

A *file* will not be packed if:

- It appears to already be packed.
- It has links.
- It is a directory.
- It cannot be opened.
- It is empty.
- The packed file will not be smaller than the original.
- Adding the **.z** extension would make the filename more than {NAME_MAX} bytes long.
- The file *file***.z** already exists.
- The **.z** file cannot be created.
- An I/O error occurred during processor.

Pack takes the following options:

–f Force packing of files.

– If one of the operands is **–**, *pack* writes to standard output the number of times each byte is used, its relative frequency, and the code for that byte. Subsequent uses of **–** instead of *file* on the command line turn this feature off and on.

DIAGNOSTICS

The *pack* utility exits with a status of 0 for success and >0 if an error occurred.

NOTES

Traditional implementations of *pack* use Huffman encoding. This implementation is a link to the *gzip* program, and uses the compression algorithms available to *gzip*.

SEE ALSO

compress, *pcat*, *unpack*

NAME

paste – merge corresponding or subsequent lines of files

SYNOPSIS

`paste [-s] [-d` *`list`*`]` *`file ...`*

DESCRIPTION

The *paste* utility concatenates the corresponding lines of the given input files, replacing all but the last file's newline characters with a single tab character, and writes the resulting lines to standard output. If end-of-file is reached on an input file while other input files still contain data, the file is treated as if it were an endless source of empty lines.

The options are as follows:

–d *list* Use one or more of the provided characters to replace the newline characters instead of the default tab. The characters in *list* are used circularly, i.e., when *list* is exhausted the first character from *list* is reused. This continues until a line from the last input file (in default operation) or the last line in each file (using the **–s** option) is displayed, at which time *paste* begins selecting characters from the beginning of *list* again.

The following special characters can also be used in *list*:

\n	newline character
\t	tab character
\\	backslash character
\0	Empty string (not a null character).

Any other character preceded by a backslash is equivalent to the character itself.

–s Concatenate all of the lines of each separate input file in command line order. The newline character of every line except the last line in each input file is replaced with the tab character, unless otherwise specified by the **–d** option.

If **–** is specified for one or more of the input files, the standard input is used; standard input is read one line at a time, circularly, for each instance of **–.**

DIAGNOSTICS

The *paste* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

cut

NAME

patch – a program for applying a diff file to an original

SYNOPSIS

```
patch [-bEflNRrsTtvZ] [-c|-e|n|u] [-d dir] [-D define]
      [-i patchfile] [-o outfile] [-p number]
      [-r rejectfile] orig patchfile
       [+ [options] orig]
patch < patchfile
```

DESCRIPTION

Patch will take a patch file containing any of the three forms of difference listing produced by the *diff* program and apply those differences to an original file (*orig*), producing a patched version. By default, the patched version is put in place of the original, with the original file backed up to the same name with the extension **.orig**, or as specified by the **–b** switch. You may also specify where you want the output to go with a **–o** switch. If *patchfile* is omitted, or is a hyphen, the patch will be read from standard input.

Upon startup, patch will attempt to determine the type of the diff listing, unless over-ruled by a **–c**, **–e**, or **–n** switch. Context diffs and normal diffs are applied by the *patch* program itself, while ed diffs are simply fed to the *ed* editor via a pipe.

Patch will try to skip any leading garbage, apply the diff, and then skip any trailing garbage. Thus you could feed an article or message containing a diff listing to *patch*, and it should work. If the entire diff is indented by a consistent amount, this will be taken into account.

With context diffs, and to a lesser extent with normal diffs, *patch* can detect when the line numbers mentioned in the patch are incorrect, and will attempt to find the correct place to apply each hunk of the patch. As a first guess, it takes the line number mentioned for the hunk, plus or minus any offset used in applying the previous hunk. If that is not the correct place, *patch* will scan both forwards and backwards for a set of lines matching the context given in the hunk. First *patch* looks for a place where all lines of the context match. If no such place is found, and it's a context diff, and the maximum fuzz factor is set to 1 or more, then another scan takes place ignoring the first and last line of context. If that fails, and the maximum fuzz factor is set to 2 or more, the first two and last two lines of context are ignored, and another scan is made. (The default maximum fuzz factor is 2.) If *patch* cannot find a place to install that hunk of the patch, it will put the hunk out to a reject file, which normally is the name of the output file plus **.rej**. (Note that the rejected hunk will come out in context diff form whether the input patch was a context diff or a normal diff. If the input was a normal diff, many of the contexts will simply be null.) The line numbers on the hunks in the reject file may be different than in the patch file: they reflect the approximate location patch thinks the failed hunks belong in the new file rather than the old one.

As each hunk is completed, you will be told whether the hunk succeeded or failed, and which line (in the new file) *patch* thought the hunk should go on. If this is different from the line number specified in the diff you will be told the offset. A single large offset *may* be an indication that a hunk was installed in

the wrong place. You will also be told if a fuzz factor was used to make the match, in which case you should also be slightly suspicious.

If no original file is specified on the command line, *patch* will try to figure out from the leading garbage what the name of the file to edit is. In the header of a context diff, the filename is found from lines beginning with "***" or "---", with the shortest name of an existing file winning. Only context diffs have lines like that, but if there is an "Index:" line in the leading garbage, *patch* will try to use the filename from that line. The context diff header takes precedence over an Index line. If no filename can be intuited from the leading garbage, you will be asked for the name of the file to patch.

(If the original file cannot be found, but a suitable SCCS or RCS file is handy, *patch* will attempt to get or check out the file.)

Additionally, if the leading garbage contains a "Prereq: " line, *patch* will take the first word from the prerequisites line (normally a version number) and check the input file to see if that word can be found. If not, *patch* will ask for confirmation before proceeding.

The upshot of all this is that you should be able to say, while in a news interface, the following:

```
| patch -d /usr/src/local/blurfl
```

and patch a file in the **blurfl** directory directly from the article containing the patch.

If the patch file contains more than one patch, *patch* will try to apply each of them as if they came from separate patch files. This means, among other things, that it is assumed that the name of the file to patch must be determined for each diff listing, and that the garbage before each diff listing will be examined for interesting things such as filenames and revision level, as mentioned previously. You can give switches (and another original file name) for the second and subsequent patches by separating the corresponding argument lists by a '+'. (The argument list for a second or subsequent patch may not specify a new patch file, however.)

Patch recognizes the following switches:

–B *prefix*
: Prepend the specified *prefix* to the names of backup files.

–b
: Back up the original contents of the file before applying the patch.

–c
: Interpret the patch file as a context diff.

–d *dir*
: Change to directory *dir* before doing anything else.

–D *define*
: Mark changes with the "#ifdef *define*...#endif" construct. The *define* argument will be used as the differentiating symbol. Unlike the C compiler, there must be a space between the **–D** and the argument.

–E
: Remove output files that are empty after using *patch*.

–e Interpret the patch file as an *ed* script.

–f assume that the user knows exactly what he or she is doing, and to not ask any questions. It does not suppress commentary, however. Use **–s** for that.

–F*number*
Set the maximum fuzz factor to *number* for inexact matching. This switch only applied to context diffs, and causes *patch* to ignore up to that many lines in looking for places to install a hunk. Note that a larger fuzz factor increases the odds of a faulty patch. The default fuzz factor is 2, and it may not be set to more than the number of lines of context in the context diff, ordinarily 3.

–f Run in batch mode (force patches); if a patch has a bad Prereq, assume it is in the correct order (as opposed to **–t**, which assumes it is reversed).

–g *num* If *num* is positive, get files from RCS or SCCS; if *num* is negative, ask before getting files from version control.

–i *patchfile*
Read the patch information from the specified *patchfile* instead of standard input.

–l Match patterns loosely, ignoring white space changes, in case the tabs and spaces have been munged in your input file. Any sequence of whitespace in the pattern line will match any sequence in the input file. Normal characters must still match exactly. Each line of the context must still match a line in the input file.

–n Interpret the patch file as a normal diff (see also **–u**).

–N ignore patches that it thinks are reversed or already applied. See also **–R**.

–o *outfile*
Use the name *outfile* for the patched file instead of replacing the original file.

–p *number*
Strip *number* leading components from pathnames found in the patch file. The strip count specifies how many pathname components are to be stripped from the front of the pathname. For example, supposing the filename in the patch file was

```
/u/howard/src/blurfl/blurfl.c
```

setting **–p** or **–p 0** gives the entire pathname unmodified, **–p 1** gives

```
u/howard/src/blurfl/blurfl.c
```

without the leading slash, **–p 4** gives

```
blurfl/blurfl.c
```

and not specifying **–p** at all just gives you "blurfl.c". Whatever you end up with is looked for either in the current directory, or the directory specified by the **–d** switch.

–r *rejectfile*
Use *rejectfile* as the reject file name, instead of using the original file name plus the extension **.rej**.

–R Reverse the order of old and new files — that is, this patch was created with the old and new files swapped. *Patch* will attempt to swap each hunk around before applying it. Rejects will come out in the swapped format. The **–R** switch will not work with *ed* diff scripts because there is too little information to reconstruct the reverse operation.

If the first hunk of a patch fails, *patch* will reverse the hunk to see if it can be applied that way. If it can, you will be asked if you want to have the **–R** switch set. If it can't, the patch will continue to be applied normally. (Note: this method cannot detect a reversed patch if it is a normal diff and if the first command is an append (i.e. it should have been a delete) since appends always succeed, due to the fact that a null context will match anywhere. Luckily, most patches add or change lines rather than delete them, so most reversed normal diffs will begin with a delete, which will fail, triggering the heuristic.)

–s Operate silently, unless an error occurs.

–T Set times of patched files, assumign that *diff* uses local time. (See also **–Z**).

–t Run in batch mode without asking questions. If a patch has a bad Prereq, assume that the patch is reversed (see **–R**).

–u Interpret the patch as a unified difference (as opposed to **–n**).

–V *style* Use the specified *style* for version control. The *style* is one of **simple**, **numbered**, or **existing**.

–v Print out revision header and patch level of *patch*.

–x*number*
Set internal debugging flags, and is of interest only to *patch* patchers.

–Y *prefix*
Prepend the specified *prefix* to the basenames of backup files.

–Z Set times of patched files, assumign that *diff* uses UTC.

–z *suffix*
Append the specified *suffix* to the names of backup files.

ENVIRONMENT VARIABLES

The *patch* utility is affected by the following environment variables, if set:

POSIXLY_CORRECT
Controls POSIX-standard behavior. If this environment variable doesn't exist in the environment or is set to a non-zero value, *patch* behaves exactly as specified by the POSIX.2 standard. Setting this environment variable to 0 enables extensions to the standard.

FILES

$OPENNT_ROOT/tmp/patch*

Temporary files.

NOTES FOR PATCH SENDERS

There are several things you should bear in mind if you are going to be sending out patches.

First, you can save people a lot of grief by keeping a **patchlevel.h** file which is patched to increment the patch level as the first diff in the patch file you send out.

If you put a Prereq: line in with the patch, it won't let them apply patches out of order without some warning. Second, make sure you've specified the filenames right, either in a context diff header, or with an Index: line. If you are patching something in a subdirectory, be sure to tell the patch user to specify a **–p** switch as needed.

Third, you can create a file by sending out a diff that compares a null file to the file you want to create. This will only work if the file you want to create doesn't exist already in the target directory. Patches against **/dev/null** will work if *POSIXLY_CORRECT* is 0.

Fourth, take care not to send out reversed patches, since it makes people wonder whether they already applied the patch. Fifth, while you may be able to get away with putting 582 diff listings into one file, it is probably wiser to group related patches into separate files in case something goes haywire.

DIAGNOSTICS

Too many to list here, but generally indicative that *patch* couldn't parse your patch file.

The message "Hmm..." indicates that there is unprocessed text in the patch file and that *patch* is attempting to intuit whether there is a patch in that text and, if so, what kind of patch it is.

CAVEATS

Patch cannot tell if the line numbers are off in an ed script, and can only detect bad line numbers in a normal diff when it finds a "change" or a "delete" command. A context diff using fuzz factor 3 may have the same problem. Until a suitable interactive interface is added, you should probably do a context diff in these cases to see if the changes made sense. Of course, compiling without errors is a pretty good indication that the patch worked, but not always.

Patch usually produces the correct results, even when it has to do a lot of guessing. However, the results are guaranteed to be correct only when the patch is applied to exactly the same version of the file that the patch was generated from.

BUGS

Could be smarter about partial matches, excessively deviant offsets and swapped code, but that would take an extra pass.

If code has been duplicated (for instance with #ifdef OLDCODE ... #else ... #endif), *patch* is incapable of patching both versions, and, if it works at all, will likely patch the wrong one, and tell you that it succeeded to boot.

If you apply a patch you've already applied, *patch* will think it is a reversed patch, and offer to un-apply the patch. This could be construed as a feature.

SEE ALSO

diff, *ed*

NAME

pathchk – check pathnames

SYNOPSIS

`pathchk [-p] pathname ...`

DESCRIPTION

The *pathchk* utility checks the pathnames as specified by the *pathname* operand for portability problems. By default, *pathchk* checks each component of the pathname, reporting diagnostic warnings when the pathname:

- is longer than {PATH_MAX} bytes.
- contains a component longer than {NAME_MAX} bytes.
- contains a component in a directory that is not searchable.
- contains a character in any component that is not valid.

The options are as follows:

–p Instead of making the checks against the underlying filesystem (NTFS), reports on the following portability-related checks against the POSIX limits, when the pathname:

- is longer than {_POSIX_PATH_MAX} bytes.
- contains a component longer than {_POSIX_NAME_MAX} bytes.
- contains a character in any component that is not in the POSIX portable filename character set.

The *pathchk* utility tests whether a pathname *could* be created. It does not require *pathname* to exist as a file.

DIAGNOSTICS

The *pathchk* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

test

NAME

pax – read and write file archives and copy directory hierarchies

SYNOPSIS

```
pax [–cdnv] [–f archive] [–s replstr] ... [–U user]
    ... [–G group] ... [–T [from_date][,to_date]]
    ... [pattern ...]

pax –r [–cdiknuvDYZ] [–f archive] [–o options] ...
    [–p string] ... [–s replstr] ...
    [–E limit] [–U user] ... [–G group] ...
    [–T [from_date][,to_date]] ...
    [pattern ...]

pax –w [–dituvX] [–b blocksize] [[–a] [–f archive]]
    [–x format] [–s replstr] ... [–o options] ...
    [–U user] ...  [–G group] ... [–B bytes]
    [–T [from_date] [,to_date][/[c][m]]] ...
    [file ...]

pax –rw [–diklntuvDXYZ] [–p string] ...
    [–s replstr] ...  [–U user] ... [–G group] ...
    [–T [from_date] [,to_date] [/[c][m]]] ...
    [file ...] directory
```

DESCRIPTION

Pax will read, write, and list the members of an archive file (including *tar*-format archives), and will copy directory hierarchies. *Pax* operation is independent of the specific archive format, and supports a wide variety of different archive formats. A list of supported archive formats can be found under the description of the **–x** option.

If invoked as *tar*, *pax* behaves as *tar*. See the *tar* reference page.

The presence of the **–r** and the **–w** options specifies which of the following functional modes *pax* will operate under: *list*, *read*, *write*, and *copy*.

<none> *List*. *Pax* will write to standard output a table of contents of the members of the archive file read from standard input, whose pathnames match the specified *patterns*. The table of contents contains one filename per line and is written using single line buffering.

–r *Read*. *Pax* extracts the members of the archive file read from the standard input, with pathnames matching the specified *patterns*. The archive format and blocking is automatically determined on input. When an extracted file is a directory, the entire file hierarchy rooted at that directory is extracted. All extracted files are created relative to the current file hierarchy. The setting of ownership, access and modification times, and file mode of the extracted files are discussed in more detail under the **–p** option.

–w *Write. Pax* writes an archive containing the *file* operands to standard output using the specified archive format. When no *file* operands are specified, a list of files to copy with one per line is read from standard input. When a *file* operand is also a directory, the entire file hierarchy rooted at that directory will be included.

–rw *Copy*. *Pax* copies the *file* operands to the destination *directory*. When no *file* operands are specified, a list of files to copy with one per line is read from the standard input. When a *file* operand is also a directory the entire file hierarchy rooted at that directory will be included. The effect of the *copy* is as if the copied files were written to an archive file and then subsequently extracted, except that there may be hard links between the original and the copied files (see the **–l** option below).

Warning: The destination *directory* must not be one of the *file* operands or a member of a file hierarchy rooted at one of the *file* operands. The result of a *copy* under these conditions is unpredictable.

While processing a damaged archive during a *read* or *list* operation, *pax* will attempt to recover from media defects and will search through the archive to locate and process the largest number of archive members possible (see the **–E** option for more details on error handling).

OPERANDS

The *directory* operand specifies a destination directory pathname. If the *directory* operand does not exist, or it is not writeable by the user, or it is not of type directory, *Pax* will exit with a non-zero exit status.

The *pattern* operand is used to select one or more pathnames of archive members. Archive members are selected using the pattern matching notation described by *fnmatch*. When the *pattern* operand is not supplied, all members of the archive will be selected. When a *pattern* matches a directory, the entire file hierarchy rooted at that directory will be selected. When a *pattern* operand does not select at least one archive member, *pax* will write these *pattern* operands in a diagnostic message to standard error and then exit with a non-zero exit status.

The *file* operand specifies the pathname of a file to be copied or archived. When a *file* operand does not select at least one archive member, *pax* will write these *file* operand pathnames in a diagnostic message to standard error and then exit with a non-zero exit status.

OPTIONS

The following options are supported:

–a Append *files* to the end of an archive that was previously written. If an archive format is not specified with a **–x** option, the format currently being used in the archive will be selected. Any attempt to append to an archive in a format different from the format already used in the archive will cause *pax* to exit immediately with a non-zero exit status. The blocking size used in the archive volume where writing starts will continue to be used for the remainder of that archive volume.

Warning: Many storage devices are not able to support the operations necessary to perform an append operation. Any attempt to append to an archive stored on such a device may damage the archive or have other unpredictable results. An archive stored in a regular file system file or on a disk device will usually support an append operation.

–b *blocksize*
When *writing* an archive, block the output at a positive decimal integer number of bytes per write to the archive file. The *blocksize* must be a multiple of 512 bytes with a maximum of 32256 bytes. A *blocksize* can end with *k* or *b* to specify multiplication by 1024 (1K) or 512, respectively. A pair of *blocksizes* can be separated by *x* to indicate a product. A specific archive device may impose additional restrictions on the size of blocking it will support. When blocking is not specified, the default *blocksize* is dependent on the specific archive format being used (see the **–x** option).

–B *bytes* Limit the number of bytes written to a single archive volume to *bytes*. The *bytes* limit can end with **m**, **k**, or **b** to specify multiplication by 1048576 (1M), 1024 (1K) or 512, respectively. A pair of *bytes* limits can be separated by *x* to indicate a product.

Warning: Only use this option when writing an archive to a device which supports an end-of-file read condition based on last (or largest) write offset (such as a regular file or a tape drive). The use of this option with a floppy or hard disk is not recommended.

–c Match all file or archive members *except* those specified by the *pattern* and *file* operands.

–d Cause files of type directory being copied or archived, or archive members of type directory being extracted, to match only the directory file or archive member and not the file hierarchy rooted at the directory.

–D This option is the same as the **–u** option, except that the file status change time is checked instead of the file modification time. The file status change time can be used to select files whose file status information (e.g. uid, gid, etc.) is newer than a copy of the file in the destination *directory*.

–f *archive*
Specify *archive* as the pathname of the input or output archive, overriding the default standard input (for *list* and *read*) or standard output (for *write*). A single archive may span multiple files and different archive devices. When required, *pax* will prompt for the pathname of the file or device of the next volume in the archive. If this option is used, it takes precedence over the value of the *TAPE* environment variable.

–E *limit* Limit the number of consecutive read faults while trying to read a flawed archives to *limit*. With a positive *limit*, *pax* will attempt to recover from an archive read error and will continue processing starting with the next file stored in the archive. A *limit* of 0 will cause

pax to stop operation after the first read error is detected on an archive volume. A *limit* of **NONE** will cause *pax* to attempt to recover from read errors forever. The default *limit* is a small positive number of retries.

Warning: Using this option with **NONE** should be used with extreme caution as *pax* may get stuck in an infinite loop on a very badly flawed archive.

–G *group*
Select a file based on its *group* name, or when starting with a **#**, a numeric gid. A '\' can be used to escape the **#**. Multiple **–G** options may be supplied and checking stops with the first match.

–i Interactively rename files or archive members. For each archive member matching a *pattern* operand or each file matching a *file* operand, *pax* will prompt to **/dev/tty** giving the name of the file, its file mode and its modification time. *Pax* will then read a line from **/dev/tty**. If this line is blank, the file or archive member is skipped. If this line consists of a single period, the file or archive member is processed with no modification to its name. Otherwise, its name is replaced with the contents of the line. *Pax* will immediately exit with a non-zero exit status if <EOF> is encountered when reading a response or if **/dev/tty** cannot be opened for reading and writing.

–k Do not overwrite existing files.

–l Link files. (The letter ell). In the *copy* mode hard links are made between the source and destination file hierarchies whenever possible.

–n Select the first archive member that matches each *pattern* operand. No more than one archive member is matched for each *pattern*. When members of type directory are matched, the file hierarchy rooted at that directory is also matched (unless **–d** is also specified).

–o *options*
Information to modify the algorithm for extracting or writing archive files which is specific to the archive format specified by **–x**. In general, *options* take the form: *name=value*

–p *string*
Specify one or more file characteristic options (privileges). The *string* option-argument is a string specifying file characteristics to be retained or discarded on extraction. The string consists of the specification characters **a**, **e**, **m**, **o**, and **p**. Multiple characteristics can be concatenated within the same string and multiple **–p** options can be specified. The meaning of the specification characters are as follows:

a Do not preserve file access times. By default, file access times are preserved whenever possible.

e 'Preserve everything', the user ID, group ID, file mode bits, file access time, and file modification time. This is intended to be used by someone with the appropriate privileges, in order to preserve all aspects of the files as they are recorded

in the archive. The **e** flag is the sum of the **o** and **p** flags.

m Do not preserve file modification times. By default, file modification times are preserved whenever possible.

o Preserve the user ID and group ID.

p Preserve the file mode bits. This intended to be used by a *user* with regular privileges who wants to preserve all aspects of the file other than the ownership. The file times are preserved by default, but two other flags are offered to disable this and use the time of extraction instead.

In the preceding list, 'preserve' indicates that an attribute stored in the archive is given to the extracted file, subject to the permissions of the invoking process. Otherwise the attribute of the extracted file is determined as part of the normal file creation action. If neither the **e** nor the **o** specification character is specified, or the user ID and group ID are not preserved for any reason, *pax* will not set the `S_ISUID` *(setuid)* and `S_ISGID` *(setgid)* bits of the file mode. If the preservation of any of these items fails for any reason, *pax* will write a diagnostic message to standard error. Failure to preserve these items will affect the final exit status, but will not cause the extracted file to be deleted. If the file characteristic letters in any of the string option-arguments are duplicated or conflict with each other, the one(s) given last will take precedence. For example, if

```
-p eme
```

is specified, file modification times are still preserved.

–r Read an archive file from standard input and extract the specified *files*. If any intermediate directories are needed in order to extract an archive member, these directories will be created as if *mkdir* was called with the bitwise inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the mode argument. When the selected archive format supports the specification of linked files and these files cannot be linked while the archive is being extracted, *pax* will write a diagnostic message to standard error and exit with a non-zero exit status at the completion of operation.

–s *replstr*

Modify the file or archive member names specified by the *pattern* or *file* operands according to the substitution expression *replstr*, using the syntax of the *ed* utility regular expressions. The format of these regular expressions are:

```
/old/new/[gp]
```

As in *ed*, *old* is a basic regular expression and *new* can contain an ampersand (&), \n (where *n* is a digit) back-references, or subexpression matching. The *old* string may also contain newline characters. Any non-null character can be used as a delimiter (/ is shown here). Multiple **–s** expressions can be specified. The expressions are applied in the order they are specified on the command line, terminating with the first successful substitution. The

optional trailing *g* continues to apply the substitution expression to the pathname substring which starts with the first character following the end of the last successful substitution. The first unsuccessful substitution stops the operation of the *g* option. The optional trailing *p* will cause the final result of a successful substitution to be written to standard error in the following format:

```
<original pathname> >> <new pathname>
```

File or archive member names that substitute to the empty string are not selected and will be skipped.

–t Reset the access times of any file or directory read or accessed by *pax* to be the same as they were before being read or accessed by *pax*.

–T [*from_date*][**,***to_date*][**/**[**c**][**m**]]

Allow files to be selected based on a file modification or status change time falling within a specified time range of *from_date* to *to_date* (the dates are inclusive). If only a *from_date* is supplied, all files with a modification or status change time equal to or younger are selected. If only a *to_date* is supplied, all files with a modification or status change time equal to or older will be selected. When the *from_date* is equal to the *to_date*, only files with a modification or status change time of exactly that time will be selected.

When *pax* is in the *write* or *copy* mode, the optional trailing field *[c][m]* can be used to determine which file time (file status change, file modification or both) are used in the comparison. If neither is specified, the default is to use file modification time only. The *m* specifies the comparison of file modification time (the time when the file was last written). The *c* specifies the comparison of status change time (the time when the file status information was last changed; e.g. a change of owner, group, mode, etc). When *c* and *m* are both specified, then the modification and status change times are both compared. The status change time comparison is useful in selecting files whose attributes were recently changed or selecting files which were recently created and had their modification time reset to an older time (as what happens when a file is extracted from an archive and the modification time is preserved). Time comparisons using both file times is useful when *pax* is used to create a time based incremental archive (only files that were changed during a specified time range will be archived).

A time range is made up of six different fields and each field must contain two digits. The format is:

```
[yy[mm[dd[hh]]]]mm[.ss]
```

Where **yy** is the last two digits of the year, the first **mm** is the month (from 01 to 12), **dd** is the day of the month (from 01 to 31), **hh** is the hour of the day (from 00 to 23), the second **mm** is the minute (from 00 to 59), and **ss** is the seconds (from 00 to 59). The minute field **mm** is required, while the other fields are optional and must be added in the following order:

```
hh , dd , mm , yy .
```

The **ss** field may be added independently of the other fields. Time ranges are relative to the current time, so

```
-T 1234/cm
```

would select all files with a modification or status change time of 12:34 PM today or later. Multiple **–T** time range can be supplied and checking stops with the first match.

–u Ignore files that are older (having a less recent file modification time) than a pre-existing file or archive member with the same name. During *read*, an archive member with the same name as a file in the file system will be extracted if the archive member is newer than the file. During *write*, a file system member with the same name as an archive member will be written to the archive if it is newer than the archive member. During *copy*, the file in the destination hierarchy is replaced by the file in the source hierarchy or by a link to the file in the source hierarchy if the file in the source hierarchy is newer.

–U *user* Select a file based on its *user* name, or when starting with a **#**, a numeric uid. A '\' can be used to escape the **#**. Multiple **–U** options may be supplied and checking stops with the first match.

–v During a *list* operation, produce a verbose table of contents using the format of the *ls* utility with the **–l** option. For pathnames representing a hard link to a previous member of the archive, the output has the format:

```
<ls -l listing> == <link name>
```

Where <*ls -l listing*> is the output format specified by the *ls* utility when used with the **–l** option. Otherwise for all the other operational modes (*read*, *write*, and *copy*), pathnames are written and flushed to standard error without a trailing newline as soon as processing begins on that file or archive member. The trailing newline, is not buffered, and is written only after the file has been read or written.

–w Write files to the standard output in the specified archive format. When no *file* operands are specified, standard input is read for a list of pathnames with one per line without any leading or trailing <blanks>.

–x *format*

Specify the output archive format, with the default format being *ustar*. *Pax* currently supports the following formats:

cpio The extended cpio interchange format specified in the POSIX.2 standard. The default blocksize for this format is 5120 bytes. Inode and device information about a file (used for detecting file hard links by this format) which may be truncated by this format is detected by *pax* and is repaired.

bcpio The old binary cpio format. The default blocksize for this format is 5120 bytes. This format is not very portable and should not be used when other formats are available. Inode and device information about a file (used for detecting file hard links by this format) which may be truncated by this format is detected by *pax* and is repaired.

sv4cpio The System V release 4 cpio. The default blocksize for this format is 5120 bytes. Inode and device information about a file (used for detecting file hard links by this format) which may be truncated by this format is detected by *pax* and is repaired.

sv4crc The System V release 4 cpio with file crc checksums. The default blocksize for this format is 5120 bytes. Inode and device information about a file (used for detecting file hard links by this format) which may be truncated by this format is detected by *pax* and is repaired.

tar The old BSD tar format as found in BSD4.3. The default blocksize for this format is 10240 bytes. Pathnames stored by this format must be 100 characters or less in length. Only regular files, hard links, and directories will be archived (other file system types are not supported). For backwards compatibility with even older tar formats, a **–o** option can be used when writing an archive to omit the storage of directories. This option takes the form:

```
-o write_opt=nodir
```

ustar The extended tar interchange format specified in the POSIX.2 standard. The default blocksize for this format is 10240 bytes. Pathnames stored by this format must be 250 characters or less in length.

Pax will detect and report any file that it is unable to store or extract as the result of any specific archive format restrictions. The individual archive formats may impose additional restrictions on use. Typical archive format restrictions include (but are not limited to): file pathname length, file size, link pathname length and the type of the file.

–X When traversing the file hierarchy specified by a pathname, do not descend into directories that have a different device ID. See the *st_dev* field as described in *stat* for more information about device ID's.

–Y This option is the same as the **–D** option, except that the file status change time is checked using the pathname created after all the file name modifications have completed.

–Z This option is the same as the **–u** option, except that the modification time is checked using the pathname created after all the file name modifications have completed.

The options that operate on the names of files or archive members (**–c**, **–D**, **–G**, **–i**, **–n**, **–s**, **–T**, **–u**, **–U**, **–v**, **–Y**, and **–Z**) interact as follows.

When extracting files during a *read* operation, archive members are 'selected', based only on the user specified pattern operands as modified by the **–c**, **–D**, **–G**, **–n**, **–T**, **–u**, and **–U** options. Then any **–s** and **–i** options will modify in that order, the names of these selected files. Then the **–Y** and **–Z** options will be applied based on the final pathname. Finally the **–v** option will write the names resulting from these modifications.

When archiving files during a *write* operation, or copying files during a *copy* operation, archive members are 'selected', based only on the user specified pathnames as modified by the **–D**, **–G**, **–n**, **–T**, **–u**, and **–U** options (the **–D** option only applies during a copy operation). Then any **–s** and **–i** options will modify in that order, the names of these selected files. Then during a *copy* operation the **–Y** and the **–Z** options will be applied based on the final pathname. Finally the **–v** option will write the names resulting from these modifications.

When one or both of the **–u** or **–D** options are specified along with the **–n** option, a file is not considered selected unless it is newer than the file to which it is compared.

EXAMPLES

The command:

```
pax -r -v -f filename
```

gives the verbose table of contents for an archive stored in *filename*.

The following commands:

```
mkdir newdir
cd olddir
pax -rw . newdir
```

will copy the entire *olddir* directory hierarchy to *newdir*.

The command:

```
pax -r -s ',^//*usr//*,,' -f a.pax
```

reads the archive **a.pax**, with all files rooted in **/usr** into the archive extracted relative to the current directory.

The command:

```
pax -rw -i . dest_dir
```

can be used to interactively select the files to copy from the current directory to **dest_dir**.

The command:

```
pax -r -pe -U andren -G bin -f a.pax
```

will extract all files from the archive **a.pax** which are owned by "andren" with group "bin" and will preserve all file permissions.

The command:

```
pax -r -w -v -Y -Z home /backup
```

will update (and list) only those files in the destination directory **/backup** which are older (less recent file status change or file modification times) than files with the same name found in the source file tree **home**.

DIAGNOSTICS

pax will exit with one of the following values:

0 All files were processed successfully.

1 An error occurred.

Whenever *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an archive, or cannot preserve the user ID, group ID, or file mode when the **–p** option is specified, a diagnostic message is written to standard error and a non-zero exit status will be returned, but processing will continue. In the case where pax cannot create a link to a file, *pax* will not create a second copy of the file.

If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may have only partially extracted a file the user wanted. Additionally, the file modes of extracted files and directories may have incorrect file bits, and the modification and access times may be wrong.

If the creation of an archive is prematurely terminated by a signal or error, *pax* may have only partially created the archive which may violate the specific archive format specification.

If while doing a *copy*, *pax* detects a file is about to overwrite itself, the file is not copied, a diagnostic message is written to standard error and when *pax* completes it will exit with a non-zero exit status.

ENVIRONMENT VARIABLES

Pax makes use of the following environment variable, if defined:

TAPE Contains the default archive name, used if the **–f** option is not specified.

NOTES

Sometimes when creating or writing to an archive in **tar**, **ustar**, **bcpio**, or **cpio** format, the following message is printed to standard error:

```
uid nnnnnnn too large to fit in header: substituting 32666
```

On traditional systems, the user ID and group ID numbers fit into 16 bits, but on INTERIX, they can be 32 bits long. The INTERIX uids and gids don't fit into the archive formats. This implementation of *pax* substitutes the value 32666 (decimal) for the original uid or gid.

Substitution isn't a problem with the **ustar**format. That format records both the numeric and character string representations of these numbers, and the string representation takes precedence over the numeric.

The **sv4cpio** and **sv4crc** formats provide 32 bits for uid and gid numbers, so no substitution is done.

SEE ALSO

tar

NAME

pcat – unpack and concatenate files

SYNOPSIS

`pcat` *`file ...`*

DESCRIPTION

The *pcat* utility unpacks files which were compressed by *pack* and writes the result to standard output.

Each *file* is expected to be a file compressed by *pack*; the *file* operand does not need to include the **.z** suffix.

The *pcat* utility will not unpack a *file* if the file does not appear to have been packed by *pack* or if the file cannot be opened.

DIAGNOSTICS

The *pcat* utility exits with 0 for success and >0 if an error occurred.

NOTES

The *pcat* utility is a link to the *gunzip* command.

SEE ALSO

compress, *pack*, *unpack*

NAME

perl - Practical Extraction and Report Language

SYNOPSIS

```
perl   [-sTuU]
       [-hv ] [-V[:configvar]]
       [-cw ] [-d[:debugger]] [-D[number/list]]
       [-pna ] [-Fpattern ] [-l[octal]] [-0[octal]]
       [-Idir ] [-m[-]module ] [-M[-]'module...']
       [-P]
       [-S]
       [-x[dir]]
       [-i[extension]]
       [-e 'command' ] [--] [programfile] [argument]...
```

DESCRIPTION

For ease of access, the Perl manual has been split up into a number of sections. These can be viewed with the command *perldoc* (if your system does not have troff installed, use **perldoc -t**):

perl	Perl overview (this section)
perltoc	Perl documentation table of contents
perldata	Perl data structures
perlsyn	Perl syntax
perlop	Perl operators and precedence
perlre	Perl regular expressions
perlrun	Perl execution and options
perlfunc	Perl builtin functions
perlvar	Perl predefined variables
perlsub	Perl subroutines
perlmod	Perl modules
perlref	Perl references
perldsc	Perl data structures intro
perllol	Perl data structures: lists of lists
perlobj	Perl objects
perltie	Perl objects hidden behind simple variables
perlbot	Perl OO tricks and examples
perldebug	Perl debugging
perldiag	Perl diagnostic messages
perlform	Perl formats
perlipc	Perl interprocess communication
perlsec	Perl security
perltrap	Perl traps for the unwary
perlstyle	Perl style guide
perlxs	Perl XS application programming interface
perlxstut	Perl XS tutorial

perlguts	Perl internal functions for those doing extensions
perlcall	Perl calling conventions from C
perlembed	Perl how to embed perl in your C or C++ app
perlpod	Perl plain old documentation
perlbook	Perl book information

(If you're intending to read these straight through for the first time, the suggested order will tend to reduce the number of forward references.)

Additional documentation for Perl modules is available in the **$OPENNT_ROOT/usr/share/man/** directory. Some of this is distributed standard with Perl, but you'll also find third-party modules there. You should be able to view this with your *man* program by including the proper directories in the appropriate start-up files. To find out where these are, type:

```
perl -le 'use Config; print "@Config{man1dir,man3dir}"'
```

If the directories were **$OPENNT_ROOT/usr/local/man/cat1** and **$OPENNT_ROOT/usr/local/man/cat3**, you would only need to add **$OPENNT_ROOT/usr/local/man** to your *MANPATH*. If they are different, you'll have to add both stems.

If that doesn't work for some reason, you can still use the supplied **perldoc** script to view module information. You might also look into getting a replacement *man* program.

If something strange has gone wrong with your program and you're not sure where you should look for help, try the **–w** switch first. It will often point out exactly where the trouble is.

DESCRIPTION

Perl is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

Upon startup, Perl looks for your script in one of the following places:

1. Specified line by line via **–e** switches on the command line.
2. Contained in the file specified by the first filename on the command line. (Note that systems supporting the #! notation invoke interpreters this way.)
3. Passed in implicitly via standard input. This only works if there are no filename arguments—to pass arguments to a STDIN script you must explicitly specify a "-" for the script name.

With methods 2 and 3, Perl starts parsing the input file from the beginning, unless you've specified a **–x** switch, in which case it scans for the first line starting with #! and containing the word "perl", and starts there instead. This is useful for running a script embedded in a larger message. (In this case you would indicate the end of the script using the __END__ token.)

As of Perl 5, the #! line is always examined for switches as the line is being parsed. Thus, if you're on a machine that only allows one argument with the #! line, or worse, doesn't even recognize the #! line, you still can get consistent

switch behavior regardless of how Perl was invoked, even if **–x** was used to find the beginning of the script.

Because many operating systems silently chop off kernel interpretation of the #! line after 32 characters, some switches may be passed in on the command line, and some may not; you could even get a "-" without its letter, if you're not careful. You probably want to make sure that all your switches fall either before or after that 32 character boundary. Most switches don't actually care if they're processed redundantly, but getting a - instead of a complete switch could cause Perl to try to execute standard input instead of your script. And a partial **–I** switch could also cause odd results.

Parsing of the #! switches starts wherever "perl" is mentioned in the line. The sequences "-*" and "- " are specifically ignored so that you could, if you were so inclined, say

```
#!/bin/sh -- # -*- perl -*- -p
eval 'exec perl $0 -S ${1+"$@"}'
   if 0;
```

to let Perl see the **–p** switch.

If the #! line does not contain the word "perl", the program named after the #! is executed instead of the Perl interpreter. This is slightly bizarre, but it helps people on machines that don't do #!, because they can tell a program that their *SHELL* is **/usr/bin/perl**, and Perl will then dispatch the program to the correct interpreter for them.

After locating your script, Perl compiles the entire script to an internal form. If there are any compilation errors, execution of the script is not attempted. (This is unlike the typical shell script, which might run partway through before finding a syntax error.)

If the script is syntactically correct, it is executed. If the script runs off the end without hitting an exit() or die() operator, an implicit **exit(0)** is provided to indicate successful completion.

Switches

A single-character switch may be combined with the following switch, if any.

```
#!/usr/bin/perl -spi.bak   # same as -s -p -i.bak
```

Switches include:

–0[*digits*] specifies the record separator (**$/**) as an octal number. If there are no digits, the null character is the separator. Other switches may precede or follow the digits. For example, if you have a version of *find* which can print filenames terminated by the null character, you can say this:

```
find . -name '*.bak' -print0 | perl -n0e unlink
```

The special value 00 will cause Perl to slurp files in paragraph mode. The value 0777 will cause Perl to slurp files whole since there is no legal character with that value.

–a turns on autosplit mode when used with a **–n** or **–p**. An implicit split command to the @F array is done as the first thing inside the implicit while loop produced by the **–n** or **–p**.

```
perl -ane 'print pop(@F), "\n";'
```

is equivalent to

```
while (<>) {
    @F = split(' ');
    print pop(@F), "\n";
}
```

An alternate delimiter may be specified using **–F**.

–c causes Perl to check the syntax of the script and then exit without executing it. Actually, it *will* execute **BEGIN**, **END**, and **use** blocks, since these are considered as occurring outside the execution of your program.

–d runs the script under the Perl debugger. See *perldebug*.

–d:*foo* runs the script under the control of a debugging or tracing module installed as Devel::foo. E.g., **–d:DProf** executes the script using the Devel::DProf profiler. See *perldebug*.

–D *number*

–D *list* sets debugging flags. To watch how it executes your script, use **–D14**. (This only works if debugging is compiled into your Perl.) Another nice value is **–D1024**, which lists your compiled syntax tree. And **–D512** displays compiled regular expressions. As an alternative specify a list of letters instead of numbers (e.g. **–D14** is equivalent to **–Dtls**):

1	p	Tokenizing and Parsing
2	s	Stack Snapshots
4	l	Label Stack Processing
8	t	Trace Execution
16	o	Operator Node Construction
32	c	String / Numeric Conversions
64	P	Print Preprocessor Command for -P
128	m	Memory Allocation
256	f	Format Processing
512	r	Regular Expression Parsing
1024	x	Syntax Tree Dump
2048	u	Tainting Checks
4096	L	Memory Leaks (not supported anymore)
8192	H	Hash Dump — usurps values()
16384	X	Scratchpad Allocation
32768	D	Cleaning Up

–e*commandline* may be used to enter one line of script. If **–e** is given, Perl will not look for a script filename in the argument list. Multiple **–e** commands may be given to build up a multi-line script. Make sure to use semicolons where you would in a normal program.

–F*pattern* specifies the pattern to split on if **–a** is also in effect. The pattern may be surrounded by / /, "" or '', otherwise it will be put in single quotes.

–h prints a summary of the options.

–i[*extension*]
specifies that files processed by the **E** construct are to be edited in-place. It does this by renaming the input file, opening the output file by the original name, and selecting that output file as the default for print() statements. The extension, if supplied, is added to the name of the old file to make a backup copy. If no extension is supplied, no backup is made. From the shell, saying

```
$ perl -p -i.bak -e "s/foo/bar/; ... "
```

is the same as using the script:

```
#!/usr/bin/perl -pi.bak
s/foo/bar/;
```

which is equivalent to

```
#!/usr/bin/perl
while (<>) {
    if ($ARGV ne $oldargv) {
        rename($ARGV, $ARGV . '.bak');
        open(ARGVOUT, ">$ARGV");
        select(ARGVOUT);
        $oldargv = $ARGV;
    }
    s/foo/bar/;
}
continue {
    print;   # this prints to original filename
}
select(STDOUT);
```

except that the **–i** form doesn't need to compare $ARGV to $oldargv to know when the filename has changed. It does, however, use ARGVOUT for the selected filehandle. Note that STDOUT is restored as the default output filehandle after the loop.

You can use **eof** without parenthesis to locate the end of each input file, in case you want to append to each file, or reset line numbering (see example in *perlfunc/eof*).

–I*directory* Directories specified by **–I** are prepended to the search path for modules (@INC), and also tells the C preprocessor where to search for include files. The C preprocessor is invoked with **–P**; by default it searches **$OPENNT_ROOT/usr/include** and **$OPENNT_ROOT/usr/lib/perl**.

–l[*octnum*] enables automatic line-ending processing. It has two effects: first, it automatically chomps the line terminator when used with **–n** or **–p**, and second, it assigns "**$**" to have the value of *octnum* so that any print statements will have that line terminator added back on. If *octnum* is omitted, sets

$** to the current value of **$/. For instance, to trim lines to 80 columns:

```
perl -lpe 'substr($_, 80) = ""'
```

Note that the assignment **$\ = $/** is done when the switch is processed, so the input record separator can be different than the output record separator if the **–l** switch is followed by a **–0** switch:

```
gnufind / -print0 | perl -ln0e 'print "found $_" if -p'
```

This sets **$** to newline and then sets **$/** to the null character.

–m [**-**]*module*

–M [**-**]*module*

–M [**-**]*'module ...'*

–[**mM**] [**-**]*module=arg***[,***arg***]...**

–m*module* executes **use** *module***();** before executing your script.

–M*module* executes **use** module**;** before executing your script. You can use quotes to add extra code after the module name, e.g.,

```
–M'module qw(foo bar)'
```

If the first character after the **–M** or **–m** is a dash (**–**) then the **use** is replaced with **no**.

A little built-in syntactic sugar means you can also say **-mmodule=foo,bar** or **-Mmodule=foo,bar** as a shortcut for

```
-M'module qw(foo bar)'
```

This avoids the need to use quotes when importing symbols. The actual code generated by **-Mmodule=foo,bar** is **use module split(/,/,q{foo,bar})**. Note that the **=** form removes the distinction between **–m** and **–M**.

–n causes Perl to assume the following loop around your script, which makes it iterate over filename arguments somewhat like *sed -n* or *awk*:

```
while (<>) {
    ...             # your script goes here
}
```

Note that the lines are not printed by default. See **–p** to have lines printed. Here is an efficient way to delete all files older than a week:

```
find . -mtime +7 -print | perl -nle 'unlink;'
```

This is faster than using the **–exec** switch of *find* because you don't have to start a process on every filename found.

BEGIN and **END** blocks may be used to capture control before or after the implicit loop, just as in *awk*.

–p causes Perl to assume the following loop around your script, which makes it iterate over filename arguments somewhat like *sed*:

```
while (<>) {
    ...             # your script goes here
} continue {
    print;
}
```

Note that the lines are printed automatically. To suppress printing use the **–n** switch. A **–p** overrides a **–n** switch.

BEGIN and **END** blocks may be used to capture control before or after the implicit loop, just as in awk.

–P causes your script to be run through the C preprocessor before compilation by Perl. (Since both comments and cpp directives begin with the # character, you should avoid starting comments with any words recognized by the C preprocessor such as "if", "else" or "define".)

–s enables some rudimentary switch parsing for switches on the command line after the script name but before any filename arguments (or before a **––**). Any switch found there is removed from @ARGV and sets the corresponding variable in the Perl script. The following script prints "true" if and only if the script is invoked with a **–xyz** switch.

```
#!/usr/bin/perl -s
if ($xyz) { print "true\n"; }
```

–S makes Perl use the PATH environment variable to search for the script (unless the name of the script starts with a slash). Typically this is used to emulate #! startup on machines that don't support #!, in the following manner:

```
#!/usr/bin/perl
eval "exec /usr/bin/perl -S $0 $*"
        if $running_under_some_shell;
```

The system ignores the first line and feeds the script to /bin/sh, which proceeds to try to execute the Perl script as a shell script. The shell executes the second line as a normal shell command, and thus starts up the Perl interpreter. On some systems $0 doesn't always contain the full pathname, so the **–S** tells Perl to search for the script if necessary. After Perl locates the script, it parses the lines and ignores them because the variable $running_under_some_shell is never true. A better construct than **$*** would be **${1+**$@"}"**,** which handles embedded spaces and such in the filenames, but doesn't work if the script is being interpreted by csh. In order to start up sh rather than csh, some systems may have to replace the #! line with a line containing just a colon, which will be politely ignored by Perl. Other systems can't control that, and need a totally devious construct that

will work under any of csh, sh or Perl, such as the following:

```
eval '(exit $?0)' && eval 'exec /usr/bin/perl -S $0 ${1+"$@"}'
& eval 'exec /usr/bin/perl -S $0 $argv:q'
   if 0;
```

–T forces "taint" checks to be turned on so you can test them. Ordinarily these checks are done only when running setuid or setgid. It's a good idea to turn them on explicitly for programs run on another's behalf, such as CGI programs. See *perl perlsec*.

–u causes Perl to dump core after compiling your script. You can then take this core dump and turn it into an executable file by using the *undump* program (not supplied). This speeds startup at the expense of some disk space (which you can minimize by stripping the executable). (Still, a "hello world" executable comes out to about 200K on my machine.) If you want to execute a portion of your script before dumping, use the dump() operator instead. Note: availability of *undump* is platform specific and may not be available for a specific port of Perl.

–U allows Perl to do unsafe operations. Currently the only "unsafe" operations are the unlinking of directories while running as superuser, and running setuid programs with fatal taint checks turned into warnings.

–v prints the version and patchlevel of your Perl executable.

–V prints summary of the major perl configuration values and the current value of @INC.

–V:*name* Prints to STDOUT the value of the named configuration variable.

–w prints warnings about identifiers that are mentioned only once, and scalar variables that are used before being set. Also warns about redefined subroutines, and references to undefined filehandles or filehandles opened readonly that you are attempting to write on. Also warns you if you use values as a number that doesn't look like numbers, using an array as though it were a scalar, if your subroutines recurse more than 100 deep, and innumerable other things. See *perl perldiag* and *perl perltrap*.

–x *directory*
tells Perl that the script is embedded in a message. Leading garbage will be discarded until the first line that starts with #! and contains the string "perl". Any meaningful switches on that line will be applied (but only one group of switches, as with normal #! processing). If a directory name is specified, Perl will switch to that directory before running the script. The **–x** switch only controls the the disposal of leading garbage. The script must be terminated with `__END__` if there is trailing garbage to be ignored (the script can process any or all of the trailing garbage via the DATA filehandle if desired).

Perl combines (in the author's opinion, anyway) some of the best features of C, *sed*, *awk*, and *sh*, so people familiar with those languages should have little difficulty with it. (Language historians will also note some vestiges of *csh*,

Pascal, and even BASIC-PLUS.) Expression syntax corresponds quite closely to C expression syntax. Unlike most utilities, Perl does not arbitrarily limit the size of your data—if you've got the memory, Perl can slurp in your whole file as a single string. Recursion is of unlimited depth. And the hash tables used by associative arrays grow as necessary to prevent degraded performance. Perl uses sophisticated pattern matching techniques to scan large amounts of data very quickly. Although optimized for scanning text, Perl can also deal with binary data, and can make dbm files look like associative arrays. Setuid Perl scripts are safer than C programs through a dataflow tracing mechanism which prevents many stupid security holes. If you have a problem that would ordinarily use *sed* or *awk* or *sh*, but it exceeds their capabilities or must run a little faster, and you don't want to write the silly thing in C, then Perl may be for you. There are also translators to turn your *sed* and *awk* scripts into Perl scripts.

But wait, there's more...

Perl version 5 is nearly a complete rewrite, and provides the following additional benefits:

- Many usability enhancements

  It is now possible to write much more readable Perl code (even within regular expressions). Formerly cryptic variable names can be replaced by mnemonic identifiers. Error messages are more informative, and the optional warnings will catch many of the mistakes a novice might make. This cannot be stressed enough. Whenever you get mysterious behavior, try the **–w** switch!!! Whenever you don't get mysterious behavior, try using **–w** anyway.

- Simplified grammar

  The new yacc grammar is one half the size of the old one. Many of the arbitrary grammar rules have been regularized. The number of reserved words has been cut by 2/3. Despite this, nearly all old Perl scripts will continue to work unchanged.

- Lexical scoping

  Perl variables may now be declared within a lexical scope, like "auto" variables in C. Not only is this more efficient, but it contributes to better privacy for "programming in the large".

- Arbitrarily nested data structures

  Any scalar value, including any array element, may now contain a reference to any other variable or subroutine. You can easily create anonymous variables and subroutines. Perl manages your reference counts for you.

- Modularity and reusability

  The Perl library is now defined in terms of modules which can be easily shared among various packages. A package may choose to import all or a portion of a module's published interface. Pragmas (that is, compiler directives) are defined and used by the same mechanism.

- Object-oriented programming

A package can function as a class. Dynamic multiple inheritance and virtual methods are supported in a straightforward manner and with very little new syntax. Filehandles may now be treated as objects.

- Embeddable and Extensible

  Perl may now be embedded easily in your C or C++ application, and can either call or be called by your routines through a documented interface. The XS preprocessor is provided to make it easy to glue your C or C++ routines into Perl. Dynamic loading of modules is supported.

- POSIX compliant

  A major new module is the POSIX module, which provides access to all available POSIX routines and definitions, via object classes where appropriate.

- Package constructors and destructors

  The new BEGIN and END blocks provide means to capture control as a package is being compiled, and after the program exits. As a degenerate case they work just like awk's BEGIN and END when you use the **–p** or **–n** switches.

- Multiple simultaneous DBM implementations

  A Perl program may now access DBM, NDBM, SDBM, GDBM, and Berkeley DB files from the same script simultaneously. In fact, the old dbmopen interface has been generalized to allow any variable to be tied to an object class which defines its access methods.

- Subroutine definitions may now be autoloaded

  In fact, the AUTOLOAD mechanism also allows you to define any arbitrary semantics for undefined subroutine calls. It's not just for autoloading.

- Regular expression enhancements

  You can now specify non-greedy quantifiers. You can now do grouping without creating a backreference. You can now write regular expressions with embedded whitespace and comments for readability. A consistent extensibility mechanism has been added that is upwardly compatible with all old regular expressions.

Ok, that's *definitely* enough hype.

ENVIRONMENT

HOME Used if chdir has no argument.

LOGDIR
: Used if chdir has no argument and HOME is not set.

PATH Used in executing subprocesses, and in finding the script if **–S** is used.

PERL5LIB
: A colon-separated list of directories in which to look for Perl library files before looking in the standard library and the current directory. If PERL5LIB is not defined, PERLLIB is used. When running taint checks (because the script was running setuid or setgid, or the **–T** switch was used), neither variable is used. The script should instead say

```
use lib "/my/directory";
```

PERL5DB

The command used to get the debugger code. If unset, uses

```
BEGIN { require 'perl5db.pl' }
```

PERLLIB

A colon-separated list of directories in which to look for Perl library files before looking in the standard library and the current directory. If PERL5LIB is defined, PERLLIB is not used.

Apart from these, Perl uses no other environment variables, except to make them available to the script being executed, and to child processes. However, scripts running setuid would do well to execute the following lines before doing anything else, just to keep people honest:

```
$ENV{'PATH'} = '/bin:/usr/bin';    # or whatever you need
$ENV{'SHELL'} = '/bin/sh' if defined $ENV{'SHELL'};
$ENV{'IFS'} = ''          if defined $ENV{'IFS'};
```

AUTHOR

Larry Wall lwall@sems.com, with the help of oodles of other folks.

FILES

$OPENNT_ROOT/tmp/perl-e$$

temporary file for -e commands

locations

SEE ALSO

a2p (*awk* to *perl* translator), *s2p* (*sed* to *perl* translator)

DIAGNOSTICS

The **–w** switch produces some lovely diagnostics.

See *perldiag* for explanations of all Perl's diagnostics.

Compilation errors will tell you the line number of the error, with an indication of the next token or token type that was to be examined. (In the case of a script passed to Perl via **–e** switches, each **–e** is counted as one line.)

Setuid scripts have additional constraints that can produce error messages such as "Insecure dependency". See *perlsec*.

Did we mention that you should definitely consider using the **–w** switch?

BUGS

The **–w** switch is not mandatory.

Perl is at the mercy of your machine's definitions of various operations such as type casting, *atof*( ) and *sprintf*( ). The latter can even trigger a coredump when passed ludicrous input values.

If your stdio requires a seek or eof between reads and writes on a particular stream, so does Perl. (This doesn't apply to *sysread*( ) and *syswrite*( ).)

While none of the built-in data types have any arbitrary size limits (apart from memory size), there are still a few arbitrary limits: a given identifier may not be longer than 255 characters, and no component of your *PATH* may be longer

than 255 if you use **–S**. A regular expression may not compile to more than 32767 bytes internally.

See the perl bugs database at *http://perl.com/perl/bugs/*. You may mail your bug reports (be sure to include full configuration information as output by the myconfig program in the perl source tree) to *perlbug@perl.com*. If you've succeeded in compiling perl, the perlbug script in the **utils/** subdirectory can be used to help mail in a bug report.

Perl actually stands for Pathologically Eclectic Rubbish Lister, but don't tell anyone I said that.

NOTES

The Perl motto is "There's more than one way to do it." Divining how many more is left as an exercise to the reader.

The three principal virtues of a programmer are Laziness, Impatience, and Hubris. See the Camel Book for why.

NAME

pg – simple pager utility

SYNOPSIS

pg **[**–n *lines***]** **[***file* ...**]**

DESCRIPTION

The *pg* command is a simple pager for reading files. If no *file* is specified, *pg* reads the standard input. It accepts a very few commands, listed after the options.

Pg takes the following options:

–n *lines* Display *lines* lines at a time.

This *pg* implementation is very simple. It accepts the following commands while displaying a file; commands must be followed by ENTER:

?, **h**, **H**	Display a help messages.
q	Quit displaying current file and go on to the next file.
Q	Quit entirely.
ENTER	Any other command (or simply ENTER) displays the next screen of text.

DIAGNOSTICS

Pg exits with the value 0 on success and >0 on failure.

SEE ALSO

more

NAME

ping – send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

```
ping [-dfnqrvR] [-c count] [-i wait]
     [-l preload] [-p pattern] [-s packetsize]
```

DESCRIPTION

Ping uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a "struct timeval" and then an arbitrary number of "pad" bytes used to fill out the packet. The options are as follows:

–c *count*
: Stop after sending (and receiving) *count* ECHO_RESPONSE packets.

–d
: Set the SO_DEBUG option on the socket being used.

–f
: Flood ping. Outputs packets as fast as they come back or one hundred times per second, whichever is more. For every ECHO_REQUEST sent a period "." is printed, while for every ECHO_REPLY received a backspace is printed. This provides a rapid display of how many packets are being dropped. Only a user with appropriate privileges can use this option. **This can be very hard on a network and should be used with caution.**

–i *wait*
: Wait *wait* seconds *between* sending each packet. The default is to wait for one second between each packet. This option is incompatible with the **–f** option.

–l *preload*
: If *preload* is specified, *ping* sends that many packets as fast as possible before falling into its normal mode of behavior.

–n
: Numeric output only. No attempt will be made to lookup symbolic names for host addresses.

–p *pattern*
: You may specify up to 16 "pad" bytes to fill out the packet you send. This is useful for diagnosing data-dependent problems in a network. For example, **–p ff** will cause the sent packet to be filled with all ones.

–q
: Quiet output. Nothing is displayed except the summary lines at startup time and when finished.

–R
: Record route. Includes the RECORD_ROUTE option in the ECHO_REQUEST packet and displays the route buffer on returned packets. Note that the IP header is only large enough for nine such routes. Many hosts ignore or discard this option.

–r
: Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (e.g., after the interface was

dropped by *routed*).

–s *packetsize*
Specifies the number of data bytes to be sent. The default is 56, which translates into 64 ICMP data bytes when combined with the 8 bytes of ICMP header data.

–v Verbose output. ICMP packets other than ECHO_RESPONSE that are received are listed.

When using *ping* for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged". Round-trip times and packet loss statistics are computed. If duplicate packets are received, they are not included in the packet loss calculation, although the round trip time of these packets is used in calculating the minimum / average / maximum round-trip time numbers. When the specified number of packets have been sent (and received) or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. Because of the load it can impose on the network, it is unwise to use *ping* during normal operations or from automated scripts.

ICMP Packet Details

An IP header without options is 20 bytes. An ICMP ECHO_REQUEST packet contains an additional 8 bytes worth of ICMP header followed by an arbitrary amount of data. When a *packetsize* is given, this indicated the size of this extra piece of data (the default is 56). Thus the amount of data received inside of an IP packet of type ICMP ECHO_REPLY will always be 8 bytes more than the requested data space (the ICMP header).

If the data space is at least eight bytes large, *ping* uses the first eight bytes of this space to include a timestamp which it uses in the computation of round trip times. If less than eight bytes of pad are specified, no round trip times are given.

Duplicate And Damaged Packets

Ping will report duplicate and damaged packets. Duplicate packets should never occur, and seem to be caused by inappropriate link-level retransmissions. Duplicates may occur in many situations and are rarely (if ever) a good sign, although the presence of low levels of duplicates may not always be cause for alarm.

Damaged packets are obviously serious cause for alarm and often indicate broken hardware somewhere in the *ping* packet's path (in the network or in the hosts).

Trying Different Data Patterns

The (inter)network layer should never treat packets differently depending on the data contained in the data portion. Unfortunately, data-dependent problems have been known to sneak into networks and remain undetected for long periods of time. In many cases the particular pattern that will have problems is something that doesn't have sufficient "transitions", such as all ones or all zeros, or a pattern right at the edge, such as almost all zeros. It isn't necessarily enough to specify a data pattern of all zeros (for example) on the command line because the pattern that is of interest is at the data link level, and the relationship between what you type and what the controllers transmit can be complicated.

This means that if you have a data-dependent problem you will probably have to do a lot of testing to find it. If you are lucky, you may manage to find a file that either can't be sent across your network or that takes much longer to transfer than other similar length files. You can then examine this file for repeated patterns that you can test using the **–p** option of *ping*.

TTL DETAILS

The TTL value of an IP packet represents the maximum number of IP routers that the packet can go through before being thrown away. In current practice you can expect each router in the Internet to decrement the TTL field by exactly one.

The TCP/IP specification states that the TTL field for TCP packets should be set to 60, but many systems use smaller values (4.3 BSD uses 30, 4.2 used 15).

The maximum possible value of this field is 255, and most systems set the TTL field of ICMP ECHO_REQUEST packets to 255. This is why you will find you can "ping" some hosts, but not reach them with *telnet* or *ftp*.

In normal operation ping prints the ttl value from the packet it receives. When a remote system receives a ping packet, it can do one of three things with the TTL field in its response:

- Not change it; this is what Berkeley systems did before the BSD 4.3 tahoe release. In this case the TTL value in the received packet will be 255 minus the number of routers in the round-trip path.
- Set it to 255; this is what current Berkeley systems do. In this case the TTL value in the received packet will be 255 minus the number of routers in the path *from* the remote system *to* the *ping*ing host.
- Set it to some other value. Some machines use the same value for ICMP packets that they use for TCP packets, for example either 30 or 60. Others may use completely wild values.

NOTES

Many Hosts and Gateways ignore the RECORD_ROUTE option.

The maximum IP header length is too small for options like RECORD_ROUTE to be completely useful. There's not much that that can be done about this, however.

Flood pinging is not recommended in general, and flood pinging the broadcast address should only be done under very controlled conditions.

INTERIX doesn't provide a *netstat* command to go with *ping*, but a *netstat.exe* is provided with Windows NT.

NAME

pr – print files

SYNOPSIS

```
pr [+page] [–column] [–adFmrt] [[–e] [char] [gap]]
   [–h header] [[–i] [char] [gap]] [–l lines] [–o offset]
   [[–s] [char]] [[–n] [char] [width]] [–w width]
   [–] [file ...]
```

DESCRIPTION

The *pr* utility is a printing and pagination filter for text files. When multiple input files are specified, each is read, formatted, and written to standard output. By default, the input is separated into 66-line pages, each with

- A 5-line header with the page number, date, time, and the pathname of the file.
- A 5-line trailer consisting of blank lines.

If standard output is associated with a terminal, diagnostic messages are suppressed until the *pr* utility has completed processing.

When multiple column output is specified, text columns are of equal width. By default text columns are separated by at least one *blank*. Input lines that do not fit into a text column are truncated. Lines are not truncated under single column output.

In the following option descriptions, column, lines, offset, page, and width are positive decimal integers and gap is a nonnegative decimal integer.

+*page* Begin output at page number *page* of the formatted input.

– *column*
Produce output that is *column*s wide (default is 1) that is written vertically down each column in the order in which the text is received from the input file. The options **–e** and **–i** are assumed. This option should not be used with **–m**. When used with **–t**, the minimum number of lines is used to display the output.

–a Modify the effect of the **–column** option so that the columns are filled across the page in a round-robin order (e.g., when column is 2, the first input line heads column 1, the second heads column 2, the third is the second line in column 1, etc.). This option requires the use of the **–column** option.

–d Produce output that is double spaced. An extra *newline* character is output following every newline found in the input.

–e [*char*][*gap*]
Expand each input tab to the next greater column position specified by the formula *n*gap+1*, where *n* is an integer > 0. If *gap* is zero or is omitted the default is 8. All tab characters in the input are expanded into the appropriate number of spaces. If any nondigit character, *char*, is specified, it is used as the input tab character.

–F Use a *form-feed* character for new pages, instead of the default behavior that uses a sequence of *newline* characters.

–h *header*
header Use the string *header* to replace the *file*name in the header line.

–i [*char*][*gap*]
In output, replace multiple spaces with tabs whenever two or more adjacent spaces reach column positions *gap+1*, *2*gap+1*, etc. If *gap* is zero or omitted, default tab settings at every eighth column position is used. If any nondigit character, *char*, is specified, it is used as the output tab character.

–l *lines* Override the 66 line default and reset the page length to *lines*. If *lines* is not greater than the sum of both the header and trailer depths (in lines), the *pr* utility suppresses output of both the header and trailer, as if the **–t** option were in effect.

–m Merge the contents of multiple files. One line from each file specified by a file operand is written side by side into text columns of equal fixed widths, in terms of the number of column positions. The number of text columns depends on the number of file operands successfully opened. The maximum number of files merged depends on page width and the per process open file limit. The options **–e** and **–i** are assumed.

–n [*char*][*width*]
Provide *width* digit line numbering. The default for *width*, if not specified, is 5. The number occupies the first *width* column positions of each text column or each line of **–m** output. If *char* (any nondigit character) is given, it is appended to the line number to separate it from whatever follows. The default for *char* is a tab. Line numbers longer than *width* columns are truncated.

–o *offset* Each line of output is preceded by *offset* space characters. If the **–o** option is not specified, the default is zero. The space taken is in addition to the output line width.

–r Write no diagnostic reports on failure to open a file.

–s *char* Separate text columns by the single character *char* instead of by the appropriate number of spaces (default for *char* is the tab character).

–t Print neither the five-line identifying header nor the five-line trailer usually supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

–w *width*
Set the width of the line to *width* column positions for multiple text-column output only. If the **–w** option is not specified and the **–s** option is not specified, the default width is 72. If the **–w** option is not specified and the **–s** option is specified, the default width is 512.

The *file* is the pathname of a file to be printed. If no *file* operands are specified, or if a *file* operand is '-', the standard input is used. The standard input is used only if no *file* operands are specified, or if a *file* operand is '-'.

The **–s** option does not allow the option letter to be separated from its argument, and the options **–e**, **–i**, and **–n** require that both arguments, if present, not be separated from the option letter.

ERRORS

If *pr* receives an interrupt while printing to a terminal, it flushes all

accumulated error messages to the screen before terminating.

DIAGNOSTICS

The *pr* utility exits 0 on success, and 1 if an error occurs.

Error messages are written to standard error during the printing process (if output is redirected) or after all successful file printing is complete (when printing to a terminal).

NOTES

The *pr* utility does not interpret form-feed characters; a form-feed character in the text will not cause *pr* to create a new page.

SEE ALSO

cat, *pg*

NAME

printf – formatted output

SYNOPSIS

`printf` *`format`* **[***`arguments ...`***]**

DESCRIPTION

Printf takes its first argument as a format, and uses that *format* for printing all remaining arguments. The *format* is a character string which contains three types of objects: plain characters, which are simply copied to standard output, character escape sequences which are converted and copied to the standard output, and format specifications, each of which causes printing of the next successive *argument*.

The *arguments* after the first are treated as strings if the corresponding format is either **c** or **s**; otherwise it is evaluated as a C constant, with the following extensions:

- A leading plus or minus sign is allowed.
- If the leading character is a single or double quote, or not a digit, plus, or minus sign, the value is the ASCII code of the next character.

The format string is reused as often as necessary to satisfy the *arguments*. Any extra format specifications are evaluated with zero or the null string.

Character escape sequences are in backslash notation as defined in the ISO or ANSI C Standard. The characters and their meanings are as follows:

\a	Write a <bell> character.
\b	Write a <backspace> character.
\f	Write a <form-feed> character.
\n	Write a <new-line> character.
\r	Write a <carriage return> character.
\t	Write a <tab> character.
\v	Write a <vertical tab> character.
\´	Write a <single quote> character.
****	Write a backslash character.
ddd	Write an 8-bit character whose ASCII value is the 1-, 2-, or 3-digit octal number *ddd*.

Each format specification is introduced by the percent character (‘‘%’’). The remainder of the format specification includes, in the following order:

Zero or more of the following flags:

#	A ‘#’ character specifying that the value should be printed in an ‘‘alternate form’’. For **c**, **d**, and **s**, formats, this option has no effect. For the **o** formats the precision of the number is increased to force the first character of the output string to a zero. For the **x** (X) format, a non-zero result has the string 0x (0X) prepended to it. For **e**, **E**, **f**, **g**, and **G**, formats, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point only appears in the

results of those formats if a digit follows the decimal point). For **g** and **G** formats, trailing zeros are not removed from the result as they would otherwise be;

– A minus sign '–' which specifies *left adjustment* of the output in the indicated field;

\+ A '+' character specifying that there should always be a sign placed before the number when using signed formats.

' ' A space specifying that a blank should be left before a positive number for a signed format. A '+' overrides a space if both are used;

0 A zero '0' character indicating that zero-padding should be used rather than blank-padding. A '–' overrides a '0' if both are used;

Field Width:
An optional digit string specifying a *field width*; if the output string has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width (note that a leading zero is a flag, but an embedded zero is part of a field width);

Precision:
An optional period, '.', followed by an optional digit string giving a *precision* which specifies the number of digits to appear after the decimal point, for **e** and **f** formats, or the maximum number of characters to be printed from a string; if the digit string is missing, the precision is treated as zero;

Format: A character which indicates the type of format to use (one of diouxXfwEgGcs**)**.

A field width or precision may be '*' instead of a digit string. In this case an *argument* supplies the field width or precision.

The format characters and their meanings are:

diouXx The *argument* is printed as a signed decimal (d or i), unsigned decimal, unsigned octal, or unsigned hexadecimal (X or x), respectively. The *argument* is printed in the style '[–]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

eE The *argument* is printed in the style **e** where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced. An upper-case E is used for an 'E' format.

gG The *argument* is printed in style **f** or in style **e** (E) whichever gives full precision in minimum space.

c The first character of *argument* is printed.

s Characters from the string *argument* are printed until the end is reached or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing, all characters in the string are printed.

% Print a '%'; no argument is used.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width.

DIAGNOSTICS

Printf exits 0 on success, 1 on failure.

NOTES

Since the floating point numbers are translated from ASCII to floating-point and then back again, floating-point precision may be lost.

ANSI hexadecimal character constants were deliberately not provided.

SEE ALSO

echo

NAME

protoize, unprotoize – convert C source between K&R and ANSI C

SYNOPSIS

```
protoize [-CgklnNqvV] [-B dir] [-c compile_opts]
         [-d dirlist] [-p compiler] [-x dirlist]
         [file ...]

unprotoize [-knNqvV] [-c compile_opts] [-d dirlist]
           [-i indentstring] [-p compiler] [-x dirlist]
           [file ...]
```

DESCRIPTION

The *protoize* and *unprotoize* utilities convert C source files between the "traditional" (K&R) formats and the ANSI/ISO C standard format (including prototypes). Both tools make use of the *gcc* compiler, which must be installed on your system.

Protoize converts existing function declarations and definitions into prototype form and it can insert declarations and prototypes for functions which are implicitly declared. *Unprotoize* reverses the process.

The *file*s must be source files, not header files. The utilities compile the files to find the functions they define; information about the functions in *file* is saved in the file *file***.X**; the **.X** files are normally deleted (but see **–x**). The converted files replace the original files, which are renamed with the file extension **.save**. If the save file already exists, the source file is discarded.

Protoize and *unprotoize* take the following options:

–B *dir* Look in directory *dir* for the **SYSCALLS.c.X** file.

–C Rename the files to end in **.C** instead of **.c**. This can be useful when converting C++ files, but will not work properly on FAT filesystems or on filesystems available across a network, where case-sensitivity is not available. This option is available in *protoize* only.

–c *compile_opts* Pass *compile_opts* to *gcc* when creating the **.X** file. The options must be passed as a single argument, and the options **–g**, **–O**, **–c**, **–S**, and **–o** are ignored if they are part of *compile_opts*.

–d *dirlist* Search the directories in *dirlist* for files to be converted. Files in those directories named with the **–x** option will *not* be converted.

–g Add a global declarations for each function that is called and not declared. The declarations are inserted at the beginning of the file before the first function definition that contains a call to an undeclared function. This option is for *protoize* only.

–i *indentstring* Indent the old-style parameter declarations with the specified *indentstring*. This option is for *unprotoize* only.

–k Keep the **.X** files; normally they are deleted.

–l Add local declarations for each function that is called and not declared. A prototype declaration are inserted in each block which calls the function without a declaration. This option is for *protoize* only.

–N Do not create **.save** files; delete the originals. This is a dangerous option to use.

–n Do not change the files; instead, print information about the covnersions that would have been done.

–p *compiler* Use the specified *compiler* instead of *gcc*. This is intended for use with alternate versions of *gcc*.

–q Quiet mode; suppress warnings about definitions and declarations that cannot be converted.

–V Display version number.

–v Synonym for **–V**; display the version number of *protoize* or *unprotoize*.

–x *filelist* Exclude the files in *filelist* from conversion.

DIAGNOSTICS

The *protoize* utility exits with status 0 for success, and >0 if an error occurred.

COPYRIGHT

Note that this document discusses only the INTERIX versions of *protoize* and *unprotoize*; it is copyright © by Softway Systems, Inc. The *protoize* and *unprotoize* utilities are covered under the General Public License and are copyright by the Free Software Foundation.

FILES

The *protoize* utility makes use of the following file:

$OPENNT_ROOT/usr/contrib/lib/gcc-lib/*platform***/cygnus-2.7.2-970404/SYSCALLS.c.X**
Information about system calls. The value of *platform* is either **i386-pc-opennt** or **alpha-pc-opennt**, depending upon the hardware. Use the **–B** option to specify a different directory for the **SYSCALLS.c.X** file.

SEE ALSO

gcc

NAME

ps – process status

SYNOPSIS

```
ps [-aAdefhjHlwx] [-g sids] [-G grouplist] [-o format]
   [-p proclist] [-t ttylist] [-u userlist] [-U userlist]
```

DESCRIPTION

The *ps* utility displays a header line followed by lines containing information about your processes that have controlling terminals. This information is sorted by process ID.

The information displayed is selected based on a set of keywords (see the **–L –O** and **–o** options). The default output format includes, for each process, the process' ID, controlling terminal, cpu time (including both user and system time), state, and associated command.

The options are as follows:

–a Display all processes with controlling terminals, except session leaders.

–A Display all processes.

–d Display all processes, except session leaders.

–e Display all processes (same as **–A**).

–f Use full output format.

–g *sids* Display processes whose session leaders are in the list *sids*. The list is comma or blank-separated.

–G *grouplist*
Display processes whose real gids are given in *grouplist*. The list is comma or blank-separated.

–h Display help for command options.

–H Display help for output formats.

–j Include the process group ID and the session ID in the output.

–l Use long output format.

–o *format*
Specify output format; this overrides the other output format options (**–fjl**). The *format* string supplied with the **–o** option is a blank- or comma-separated list of format specifiers. Each specifier is a field name or has the form *fieldname=title*, where *title* is the string displayed in the header that *ps* prints as its first line of output. For example, **–o user=NAME** causes the user field in the output to have the title NAME.

–p *proclist*
Display processes whose process IDs are given in *proclist*. The list is comma or blank-separated.

–t *ttylist* Display processes whose ttys are given in the list *ttylist*. The list is comma or blank-separated.

–u *userlist* Display processes whose real uids are given in *userlist*. The list is comma or blank-separated.

–U *userlist* Same as **–u userlist**. The list is comma or blank-separated; if blanks are used, the list must be quoted.

–w Use 132 columns instead of 80. **–ww** means no line width limit.

–x Display processes belonging to the current user (equivalent to **ps –u $USER**). The command **ps –ax** displays all processes for the current user and is equivalent to **ps –e**. If any process-selection option other than **–a** is combined with **–x**, **–x** has no effect.

OUTPUT FORMAT

Output comes in three formats: default, full, and long; the options **–f**, **–j**, **–l**, and **–o** change the default format. The section *Format* (below) describes all of the possible output fields. Very long lines of output are truncated to 80 characters, unless either of the **–w** or **–ww** options is given.

The **default** output format displays

- the process ID,
- the name of the tty,
- the processor time used by the command,
- and the command line:

```
    PID TTY      TIME CMD
14090242 n01  0:00.03 ksh -c ps
 6553604 n01  0:00.03 ps
 8388614 n01  0:00.50 vi ps.r
```

The **full** output format displays

- the user's name,
- the process ID,
- the PID of the parent process,
- elapsed time since the process was started,
- the tty name,
- cumulative CPU time used by the process,
- and the command line:

```
  USER     PID     PPID  ELAPSED TTY     TIME CMD
juser 14155778  8388614  0:00.12 n01  0:00.01 ksh -c ps -f
juser  6619140 14155778  0:00.14 n01  0:00.02 ps -f
juser  8388614  8388608  3:23.00 n01  0:00.63 vi ps.r
```

The **long** output format displays:

- the process's state,
- the user's name,
- the process ID,

- the PID of the parent process,
- the size of the working set in kilobytes,
- elapsed time since the process was started,
- the tty name,
- cumulative CPU time used by the process,
- and the command line:

```
STATE  USER      PID     PPID    SZ  ELAPSED TTY     TIME CMD
Re    juser 14352386  8388614   584  0:00.12 n01  0:00.02 ksh -c ps -l
Re    juser  6815748 14352386   440  0:00.14 n01  0:00.05 ps -l
Re    juser  8388614  8388608  1004  6:44.08 n01  0:01.15 vi ps.r
```

State

The **–l** option provides a State field which consists of a one-letter *run state* followed by one or more *flag letters*.

The run state is one of:

R **Active**: The process is considered active; this is equivalent to a "runnable" process on traditional systems.

S **In a system call**: The process has made a request to the INTERIX subsystem, is waiting on the reply, and the subsystem is still servicing the request.

T **Stopped**: The process has been stopped.

U **Unconnected**: a freshly-created process briefly passes through this state during process initialization.

W **Waiting**: The process is blocked and waiting, either in a *wait*( ) or a *waitpid*( ) call.

Y **Tty**: Process is blocked doing tty I/O.

Z **Exiting**: This is a zombie process; the process has exited but the process' parent has not yet waited for the process.

The *flag letters* can be:

e The process has called the *exec*( ) function at some point in its history — that is, it has tried to run another process.

f The process is a Win32 process. (The **f** stands for "foreign.")

n The process has its P_NO_FORK flag set; this flag is set briefly for the period between the receipt of a signal and the response. This condition should not persist.

s The process is a session leader.

u The process or one of its ancestors has called an *exec_asuser*( ) function.

w The process has its P_WAITED flag set. This flag is normally unset. It indicates that the process was stopped and was being waited upon, but that the *wait*( ) or *waitpid*( ) call has returned. It is cleared if the process is stopped again.

Format Specifiers

The **–o** option allows you to specify the format of the output; the format string consists of field names and optional column titles for those fields.

Here are the field names, with their default titles:

Field	Title	Meaning
addr	ADDR	address (not implemented)
args	COMMAND	command and argument list
c	C	cpu use for scheduling (not implemented)
comm	COMMAND	argv[0] (the command name)
etime	ELAPSED	elapsed time
flags	F	process flags (octal)
group	GROUP	effective group ID
nice	NI	scheduling priority (not implemented)
pcpu	%CPU	cpu time used / available
pgid	PGID	process group ID
pid	PID	process ID
ppid	PPID	parent process ID
pri	PRI	priority (not implemented)
rgroup	RGROUP	real group ID
ruser	RUSER	real user ID
s	S	process state
sid	SID	session ID
state	STATE	process state & flags
stime	STIME	starting time (not implemented)
sz	SZ	working set size, in K
time	TIME	cpu time
tty	TT	controlling terminal
user	USER	effective user ID
vsz	VSZ	virtual memory size, in K
wchan	WCHAN	event blocking process (not implemented)

If you specify a field that is not implemented, *ps* displays a hyphen (-).

Since there is no way for a process to change its effective user or group ID, the **user** and **group** fields are always identical to the **ruser** and **rgroup** fields, respectively.

DIAGNOSTICS

Ps returns 0 for success and greater than 0 if an error occurred.

BUGS

Since *ps* cannot run faster than the system and is run as any other scheduled process, the information it displays can never be exact.

SEE ALSO

kill

exec(), *strftime*()

NAME

puzzle – 15-puzzle game for X

SYNOPSIS

`puzzle [-option ...]`

OPTIONS

–display *display*
This option specifies the display to use; see *X*.

–geometry *geometry*
This option specifies the size and position of the puzzle window; see *X*.

–size *WIDTHxHEIGHT*
This option specifies the size of the puzzle in squares.

–speed *num*
This option specifies the speed in tiles per second for moving tiles around.

–picture *filename*
This option specifies an image file containing the picture to use on the tiles. Try **mandrill.cm**. This only works on 8-bit pseudo-color screens.

–colormap
This option indicates that the program should create its own colormap for the picture option.

DESCRIPTION

Puzzle with no arguments plays a 4x4 15-puzzle. The control bar has two boxes in it. Clicking in the left box scrambles the puzzle. Clicking in the right box solves the puzzle. Clicking the middle button anywhere else in the control bar causes puzzle to exit. Clicking in the tiled region moves the empty spot to that location if the region you click in is in the same row or column as the empty slot.

SEE ALSO

X

BUGS

The picture option should work on a wider variety of screens.

COPYRIGHT

Copyright 1988, Don Bennett.

AUTHOR

Don Bennett, HP Labs

NAME

pwd – return working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd writes the absolute pathname of the current working directory to the standard output.

DIAGNOSTICS

The *pwd* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

cd

NAME

rcp – remote file copy

SYNOPSIS

```
rcp [–p] file1 file2
rcp [–pr] file ...  directory
```

DESCRIPTION

Rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form ‘‘rname@rhost:path’’, or a local file name (containing no ‘:’ characters, or a ‘/’ before any ‘:’s).

–p The **–p** option causes *rcp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the *umask*. By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the *umask*( ) on the destination host is used.

–r If any of the source files are directories, *rcp* copies each subtree rooted at that name; in this case the destination must be a directory.

If *path* is not a full path name, it is interpreted relative to the login directory of the specified user *ruser* on *rhost*, or your current user name if no other remote user name is specified. A *path* on a remote host may be quoted (using \, ", or ´) so that the metacharacters are interpreted remotely.

Rcp does not prompt for passwords; it performs remote execution via *rsh*, and requires the same authorization.

Rcp handles third party copies, where neither source nor target files are on the current machine.

NOTES

Doesn’t detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

Is confused by any output generated by commands in a **.login**, **.profile**, or **.cshrc** file on the remote host.

The destination user and hostname may have to be specified as ‘‘rhost.rname’’ when the destination machine is running the BSD 4.2 version of *rcp*.

FILES

The *rcp* command uses the following files, if they exist:

$HOME/.rhosts
This file is on the destination machine; see *rshd* for a description.

$HOME/.password
This file is on the destination machine; see *rshd* for a description.

SEE ALSO

cp, *ftp*, *rsh*, *rshd*, *rlogin*

NAME

rcs – change RCS file attributes

SYNOPSIS

`rcs [`*`options`*`] `*`file ...`*

DESCRIPTION

rcs creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For *rcs* to work, the caller's login name must be on the access list, except if the access list is empty, the caller is the owner of the file or the superuser, or the **–i** option is present.

Pathnames matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in *ci*. Revision numbers use the syntax described in *ci*.

Rcs accepts the following options:

–a*logins* Append the login names appearing in the comma-separated list *logins* to the access list of the RCS file.

–A*oldfile* Append the access list of *oldfile* to the access list of the RCS file.

–b[*rev*] Set the default branch to *rev*. If *rev* is omitted, the default branch is reset to the (dynamically) highest branch on the trunk.

–c*string* Set the comment leader to *string*. The comment leader is printed before every log message line generated by the keyword **Log** during checkout (see *co*). This is useful for programming languages without multi-line comments. An initial *ci*, or an **rcs –i** without **–c**, guesses the comment leader from the suffix of the working file.

–e[*logins*]
Erase the login names appearing in the comma-separated list *logins* from the access list of the RCS file. If *logins* is omitted, erase the entire access list.

–i Create and initialize a new RCS file, but do not deposit any revision. If the RCS file has no path prefix, try to place it first into the subdirectory **./RCS**, and then into the current directory. If the RCS file already exists, print an error message.

–I Run interactively, even if the standard input is not a terminal.

–k*subst* Set the default keyword substitution to *subst*. The effect of keyword substitution is described in *co*. Giving an explicit **–k** option to *co*, *rcsdiff*, and *rcsmerge* overrides this default. Beware **rcs –kv**, because **–kv** is incompatible with **co –l**. Use **rcs –kkv** to restore the normal default keyword substitution.

–l[*rev*] Lock the revision with number *rev*. If a branch is given, lock the latest revision on that branch. If *rev* is omitted, lock the latest revision on the default branch. Locking prevents overlapping changes. A lock is removed with *ci* or **rcs –u** (see below).

–L Set locking to "strict". Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for files that are shared.

–m*rev***:***msg*
Replace revision *rev*'s log message with *msg*.

–n*name*[**:**[*rev*]]
Associate the symbolic name *name* with the branch or revision *rev*. Delete the symbolic name if both **:** and *rev* are omitted; otherwise, print an error message if *name* is already associated with another number. If *rev* is symbolic, it is expanded before association. A *rev* consisting of a branch number followed by a **.** stands for the current latest revision in the branch. A **:** with an empty *rev* stands for the current latest revision on the default branch, normally the trunk. For example, **rcs –n***name***: RCS/*** associates *name* with the current latest revision of all the named RCS files; this contrasts with **rcs –n***name***:$ RCS/*** which associates *name* with the revision numbers extracted from keyword strings in the corresponding working files.

–N *name*[**:**[*rev*]]
Act like **–n**, except override any previous assignment of *name*.

–o*range* Deletes ("outdates") the revisions given by *range*. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form *rev1***:***rev2* means revisions *rev1* to *rev2* on the same branch, **:***rev* means from the beginning of the branch containing *rev* up to and including *rev*, and *rev*: means from revision *rev* to the end of the branch containing *rev*. None of the outdated revisions may have branches or locks.

–q Run quietly; do not print diagnostics.

–s*state*[:*rev*]
Set the state attribute of the revision *rev* to *state*. If *rev* is a branch number, assume the latest revision on that branch. If *rev* is omitted, assume the latest revision on the default branch. Any identifier is acceptable for *state*. A useful set of states is **Exp** (for experimental), **Stab** (for stable), and **Rel** (for released). By default, *ci* sets the state of a revision to **Exp**.

–t[*file*] Write descriptive text from the contents of the named *file* into the RCS file, deleting the existing text. The *file* pathname may not begin with **.** (dot). If *file* is omitted, obtain the text from standard input, terminated by end-of-file or by a line containing **.** by itself. Prompt for the text if interaction is possible; see **–I**. With **–i**, descriptive text is obtained even if **–t** is not given.

–t–*string*
Write descriptive text from the *string* into the RCS file, deleting the existing text.

–u[*rev*] Unlock the revision with number *rev*. If a branch is given, unlock the latest revision on that branch. If *rev* is omitted, remove the latest lock held by the caller. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated by end-of-file or by a line containing **.** by itself.

–U Set locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for checkin. This option should *not* be used for files that are shared. Whether default locking is strict is determined by your system administrator, but it is normally strict.

–V*n* Emulate RCS version *n*. See *co* for details.

–x*suffixes*
Use *suffixes* to characterize RCS files. See *ci* for details.

COMPATIBILITY

The **–b** *rev* option generates an RCS file that cannot be parsed by RCS version 3 or earlier.

The **–k** *subst* options (except **–kkv**) generate an RCS file that cannot be parsed by RCS version 4 or earlier.

Use **rcs –V***n* to make an RCS file acceptable to RCS version *n* by discarding information that would confuse version *n*.

RCS version 5.5 and earlier does not support the **–x** option, and requires a **,v** suffix on an RCS pathname.

FILES

Rcs accesses files much as *ci* does, except that it uses the effective user for all accesses, it does not write the working file or its directory, and it does not even read the working file unless a revision number of **$** is specified.

ENVIRONMENT

RCSINIT
Containst a list of options prepended to the argument list, separated by spaces. See *ci* for details.

DIAGNOSTICS

The RCS pathname and the revisions outdated are written to the diagnostic output. The exit status is zero if and only if all operations were successful.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 1.9; Release Date: 1997 / 12 / 18.
Copyright © 1982, 1988, 1989 by Walter F. Tichy.
Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

co, ci, ident, rcsdiff, rcsmerge, rlog

rcsintro, rcsfile

Walter F. Tichy, RCS—A System for Version Control, *Software—Practice & Experience* **15**, 7 (July 1985), 637-654.

BUGS

The separator for revision ranges in the **–o** option used to be **–** instead of **:**, but this leads to confusion when symbolic names contain **–.** For backwards compatibility **rcs –o** still supports the old **–** separator, but it warns about this obsolete use.

Symbolic names need not refer to existing revisions or branches. For example, the **–o** option does not remove symbolic names for the outdated revisions; you must use **–n** to remove the names.

NAME

rcsdiff – compare RCS revisions

SYNOPSIS

```
rcsdiff [-k subst] [-q ] [-r rev1 [-r rev2]]
        [-V n] [-x suffixes] [diff_options] file ...
```

DESCRIPTION

rcsdiff runs *diff* to compare two revisions of each RCS file given.

Pathnames matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in *ci*.

Rcsdiff takes all the options of *diff* that apply to regular files, with the same meaning as for *diff*. In addition, *rcsdiff* accepts:

–k*subst* Control keyword substitution when extracting revisions, as described in *co*; for example, **–kk –r1.1 –r1.2** ignores differences in keyword values when comparing revisions **1.1** and **1.2**. To avoid excess output from locker name substitution, **–kkvl** is assumed if

1. at most one revision option is given,
2. no **–k** option is given,
3. **–kkv** is the default keyword substitution, and
4. the working file's mode would be produced by **co –l**.

–q Suppress diagnostic output.

–r*rev* Specify a particular revision, *rev*. Zero, one, or two revisions may be specified with **–r**. If both *rev1* and *rev2* are omitted, *rcsdiff* compares the latest revision on the default branch (by default the trunk) with the contents of the corresponding working file. This is useful for determining what you changed since the last checkin.

If *rev1* is given, but *rev2* is omitted, *rcsdiff* compares revision *rev1* of the RCS file with the contents of the corresponding working file.

If both *rev1* and *rev2* are given, *rcsdiff* compares revisions *rev1* and *rev2* of the RCS file.

Both *rev1* and *rev2* may be given numerically or symbolically.

–V*n* Emulate RCS version *n*, where *n* may be **3**, **4**, or **5**. This may be useful when interchanging RCS files with others who are running older versions of RCS. To see which version of RCS your correspondents are running, have them invoke **rlog** on an RCS file; if none of the first few lines of output contain the string **branch**: it is version 3; if the dates' years have just two digits, it is version 4; otherwise, it is version 5. An RCS file generated while emulating version 3 will lose its default branch. An RCS revision generated while emulating version 4 or earlier will have a timestamp that is off by up to 13 hours. A revision extracted while emulating version 4 or earlier will contain dates of the form *yy/mm/dd* instead of *yyyy/mm/dd* and may also contain different white space in the substitution for **Log**.

–x*suffixes*

Use *suffixes* to characterize RCS files. See *ci* for details.

EXAMPLE

The command

```
$ rcsdiff  f.c
```

compares the latest revision on the default branch of the RCS file to the contents of the working file **f.c**.

ENVIRONMENT

RCSINIT

A list of options prepended to the argument list, separated by spaces. See *ci* for details.

DIAGNOSTICS

Exit status is 0 for no differences during any comparison, 1 for some differences, 2 for trouble.

IDENTIFICATION

Author: Walter F. Tichy.

Revision Number: 1.6; Release Date: 1997 / 11 / 12.

Copyright © 1982, 1988, 1989 by Walter F. Tichy.

Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

ci, *co*, *diff*, *ident*, *rcs*, *rcsmerge*, *rlog*

rcsintro

Walter F. Tichy, RCS—A System for Version Control, *Software—Practice & Experience* **15**, 7 (July 1985), 637-654.

NAME

rcsmerge – merge RCS revisions

SYNOPSIS

`rcsmerge [`*`options`*`] `*`file`*

DESCRIPTION

rcsmerge incorporates the changes between two revisions of an RCS file into the corresponding working file.

If *file* matches an RCS suffix it denotes an RCS file; otherwise it denotes a working file. Names are paired as explained in *ci*.

At least one revision must be specified with one of the options described below, usually **–r**. At most two revisions may be specified. If only one revision is specified, the latest revision on the default branch (normally the highest branch on the trunk) is assumed for the second revision. Revisions may be specified numerically or symbolically.

rcsmerge prints a warning if there are overlaps, and delimits the overlapping regions as explained in *merge*. The command is useful for incorporating changes into a checked-out revision.

The *rcsmerge* command takes the following options:

–k*subst* Use *subst* style keyword substitution. See *co* for details. For example, **–kk –r1.1 –r1.2** ignores differences in keyword values when merging the changes from **1.1** to **1.2**.

–p[*rev*] Send the result to standard output instead of overwriting the working file.

–q[*rev*] Run quietly; do not print diagnostics.

–r[*rev*] Merge with respect to revision *rev*. Here an empty *rev* stands for the latest revision on the default branch, normally the head.

–V*n* Emulate RCS version *n*. See *co* for details.

–x*suffixes*
Use *suffixes* to characterize RCS files. See *ci* for details.

EXAMPLES

Suppose you have released revision 2.8 of **f.c**. Assume furthermore that after you complete an unreleased revision 3.4, you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file **f.c** and execute

```
$ rcsmerge -p -r2.8 -r3.4 f.c >f.merged.c
```

Then examine **f.merged.c**. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute **co –j**:

```
$ ci -r2.8.1.1 f.c
$ co -r3.4 -j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in **f.c**.

```
$ rcsmerge  -r2.8  -r2.4  f.c"
```

Note the order of the arguments, and that **f.c** will be overwritten.

ENVIRONMENT

RCSINIT

A list of options prepended to the argument list, separated by spaces. See *ci* for details.

DIAGNOSTICS

Exit status is 0 for no overlaps, 1 for some overlaps, 2 for trouble.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 1.7; Release Date: 1997 / 11 / 12.
Copyright © 1982, 1988, 1989 by Walter F. Tichy.
Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

ci, *co*, *ident*, *rcs*, *rcsdiff*, *rlog*

Walter F. Tichy, RCS—A System for Version Control, *Software—Practice & Experience* **15**, 7 (July 1985), 637-654.

NAME

rdist – remote file distribution client program

SYNOPSIS

```
rdist [-DFn] [-A num] [-a num] [-d var=value]
      [-l local_logopts] [-L remote_logopts] [-f distfile]
      [-M maxproc] [-m host] [-o distopts] [-t timeout]
      [-p rdistd-path] [-P rsh-path] [name ...]

rdist [-DFn] [-c name ...[user@]host[:dest]

rdist -V
```

DESCRIPTION

Rdist is a program to maintain identical copies of files over multiple hosts. It preserves the owner, group, mode, and mtime of files if possible and can update programs that are executing. *Rdist* reads commands from *distfile* to direct the updating of files and / or directories. If *distfile* is '–', the standard input is used. If no **–f** option is present, the program looks first for 'distfile', then 'Distfile' to use as the input. If no names are specified on the command line, *rdist* will update all of the files and directories listed in *distfile*. Otherwise, the argument is taken to be the name of a file to be updated or the label of a command to execute. If label and file names conflict, it is assumed to be a label. These may be used together to update specific files using specific commands.

The **–c** option forces *rdist* to interpret the remaining arguments as a small *distfile*. The equivalent distfile is as follows.

```
( name ... ) -> [user@]host
        install       [dest] ;
```

Rdist uses the *rcmd*( ) function call or the *rsh*, to access each target host.

With the *rcmd*( ) method, *rdist* makes the connection to the target host itself and runs the *rdistd* server program as shown below.

On each target host *Rdist* will attempt to run the command

```
rdistd -S
```

or

```
"rdistd_path" -S
```

if the **–p** option was specified. If no **–p** option is included, or the *rdistd_path* is a simple filename, *rdistd* or *rdistd_path* must be somewhere in the *PATH* of the user running *rdist* on the remote (target) host.

OPTIONS

–A *num*

Set the minimum number of free files (inodes) on a filesystem that must exist for *rdist* to update or install a file.

–a *num*

Set the minimum amount of free space (in bytes) on a filesystem that must exist for *rdist* to update or install a file.

–D Enable copious debugging messages.

–d *var*=*value*
Define *var* to have *value*. This option is used to define or override variable definitions in the *distfile*. *Value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by tabs and / or spaces.

–F Do not fork any child *rdist* processes. All clients are updated sequentially.

–f *distfile*
Set the name of the distfile to use to be *distfile*. If *distfile* is specified as ''–'' (dash) then read from standard input (stdin).

–l *logopts*
Set local logging options. See the section "MESSAGE LOGGING" for details on the syntax for *logopts*.

–L *logopts*
Set remote logging options. *logopts* is the same as for local logging except the values are passed to the remote server (*rdistd*). See the section "MESSAGE LOGGING" for details on the syntax for *logopts*.

–M *num*
Set the maximum number of simultaneously running child *rdist* processes to *num*. The default is 4.

–m *machine*
Limit which machines are to be updated. Multiple **–m** arguments can be given to limit updates to a subset of the hosts listed in the *distfile*.

–n Print the commands without executing them. This option is useful for debugging *distfile*.

–o*distopts*
Specify the dist options to enable. *distopts* is a comma-separated list of options which are listed below. The valid values for *distopts* are:

verify Verify that the files are up to date on all the hosts. Any files that are out of date will be displayed but no files will be changed nor any mail sent.

whole Whole mode. The whole file name is appended to the destination directory name. This will preserve the directory structure of the files being copied instead of flattening the directory structure. For example, *rdist*'ing a list of files such as *//C/path/dir1/f1* and *//C/path/dir2/f2* to *//C/tmp/dir* would create files *//C/tmp/dir/path/dir1/f1* and *//C/tmp/dir/path/dir2/f2* instead of *//C/tmp/dir/dir1/f1* and *//C/tmp/dir/dir2/f2*. (Normally, only the last component of a name is used when renaming files.)

noexec Automatically exclude executable files that are in a.out format from being checked or updated. Currently not supported on INTERIX.

younger
Younger mode. Files are normally updated if their *mtime* and

size (see *stat*( )) disagree. This option causes *rdist* not to update files that are younger than the master copy. This can be used to prevent newer copies on other hosts from being replaced. A warning message is printed for files which are newer than the master copy.

compare
Binary comparison. Perform a binary comparison and update files if they differ rather than comparing dates and sizes.

follow Follow symbolic links. Copy the file that the link points to rather than the link itself. Not supported on INTERIX, since symbolic links are not supported.

ignlnks Ignore unresolved links. *Rdist* will normally try to maintain the link structure of files being transferred and warn the user if all the links cannot be found.

chknfs Do not check or update files on target host that reside on NFS filesystems.

chkreadonly
Enable check on target host to see if a file resides on a read-only filesystem. If a file does, then no checking or updating of the file is attempted.

chksym
If the target on the remote host is a symbolic link, but is not on the master host, the remote target will be left a symbolic link. This behavior is generally considered a bug in the original version of *rdist*, but is present to allow compatibility with older versions.

quiet Quiet mode. Files that are being modified are normally printed on standard output. This option suppresses this.

remove Remove extraneous files. If a directory is being updated, any files that exist on the remote host that do not exist in the master directory are removed. This is useful for maintaining truly identical copies of directories.

nochkowner
Do not check user ownership of files that already exist. The file ownership is only set when the file is updated.

nochkgroup
Do not check group ownership of files that already exist. The file ownership is only set when the file is updated.

nochkmode
Do not check file and directory permission modes. The permission mode is only set when the file is updated.

nodescend
Do not descend into a directory. Normally *rdist* will recursively check directories. If this option is enabled, then any files listed in the file list in the distfile that are directories are not recursively scanned. Only the existence, ownership, and mode of the directory are checked.

numchkgroup
Use the numeric group ID (gid) to check group ownership instead of the group name. This is useful on INTERIX when group names may contain whitespace. This is the

recommended default usage. The INTERIX group ID is essentially the Windows NT Security ID (SID); for groups other than the built-ins, it is unique.

numchkowner
Use the numeric user ID (uid) to check user ownership instead of the user name.

savetargets
Save files that are updated instead of removing them. Any target file that is updates is first rename from *file* to *file*.OLD.

sparse Enable checking for sparse (aka *wholely*) files. One of the most common types of sparse files are those produced by *ndbm*( ). This option adds some additional processing overhead so it should only be enabled for targets likely to contain sparse files.

–p *rdistd-path*
Set the path where the rdistd server is searched for on the target host.

–P *rsh-path*
Set the path to the *rsh* command. The *rsh-path* may be a colon-separated list of possible pathnames. In this case, the first component of the path to exist is used. i.e.
//C/Interix/bin/rsh:**//C/Interix/usr/sbin/remsh**:**/C/Interix/usr/bsd/rsh**

–t *timeout*
Set the timeout period (in seconds) for waiting for responses from the remote *rdist* server. The default is 900 seconds.

–V Print version information and exit.

MESSAGE LOGGING

Rdist uses a collection of predefined message *facilities* that each contain a list of message *types* specifying which types of messages to send to that *facility*. The local client (*rdist*) and the remote server (*rdistd*) each maintain their own copy of what types of messages to log to what facilities.

The **–l** *logopts* option to *rdist* tells *rdist* what logging options to use locally. The **–L** *logopts* option to *rdist* tells *rdist* what logging options to pass to the remote *rdistd* server.

The form of *logopts* should be of form

facility=*types*:*facility*=*types...*

The valid facility names are:

file Log to a file. To specify the file name, use the format

‘‘**file=***filename*=*types*’’.

e.g. **“file=//E/Interix/tmp/rdist.log=all,debug”**.

notify Use the internal *rdist* **notify** facility. This facility is used in conjunction with the **notify** keyword in a *distfile* to specify what messages are mailed to the **notify** address.

stdout Messages to standard output.

syslog Use the *syslogd* facility.

types should be a comma separated list of message types. Each message type specified enables that message level. This is unlike the *syslog*( ) system facility which uses an ascending order scheme. The following are the valid *types:*

change Things that change. This includes files that are installed or updated in some way.
info General information.
notice General info about things that change. This includes things like making directories which are needed in order to install a specific target, but which are not explicitly specified in the *distfile.*
nerror Normal errors that are not fatal.
ferror Fatal errors.
warning Warnings about errors which are not as serious as **nerror** type messages.
debug Debugging information.
all All but debug messages.

Here is a sample command line option:

```
-l stdout=all:syslog=change,notice:file=/tmp/rdist.log=all
```

This entry will set local message logging to have all but debug messages sent to standard output, change and notice messages will be sent to *syslog*( ), and all messages will be written to the file **$OPENNT_ROOT/tmp/rdist.lo**g.

DISTFILES

The *distfile* contains a sequence of entries that specify the files to be copied, the destination hosts, and what operations to perform to do the updating. Each entry has one of the following formats.

```
<variable name> '=' <name list>
[ label: ] <source list> '->' <destination list> <command list>
[ label: ] <source list> '::' <time_stamp file> <command list>
```

The first format is used for defining variables.

The second format is used for distributing files to other hosts.

The third format is used for making lists of files that have been changed since some given date.

- The *source list* specifies a list of files and / or directories on the local host which are to be used as the master copy for distribution.
- The *destination list* is the list of hosts to which these files are to be copied. Each file in the source list is added to a list of changes if the file is out of date on the host which is being updated (second format) or the file is newer than the time stamp file (third format).
- *Labels* are optional. They are used to identify a command for partial updates.

Newlines, tabs, and blanks are only used as separators and are otherwise ignored. Comments begin with '#' and end with a newline.

Variables to be expanded begin with '$' followed by one character or a name enclosed in curly braces (see the examples at the end).

The source and destination lists have the following format:

name

or

(*zero or more names separated by white-space*)

These simple lists can be modified by using one level of set addition, subtraction, or intersection like this:

list '-' list

or

list '+' list

or

list '&' list

If additional modifications are needed (e.g., ''all servers and client machines except for the OSF/1 machines'') then the list will have to be explicitly constructed in steps using "temporary" variables.

The shell meta-characters '[', ']', '{', '}', '*', and '?' are recognized and expanded (on the local host only) in the same way as *csh*. They can be escaped with a backslash. The '~' character is also expanded in the same way as *csh* but is expanded separately on the local and destination hosts. When the **–owhole** option is used with a file name that begins with '~', everything except the home directory is appended to the destination name. File names which do not begin with '/' or '~' use the destination user's home directory as the root directory for the rest of the file name.

The command list consists of zero or more commands of the following format.

install	*options*	*opt_dest_name* **;**
notify	*name list*	**;**
except	*name list*	**;**
except_pat	*pattern list*	**;**
special	*name list*	*string* **;**
cmdspecial	*name list*	*string* **;**

The **install** command is used to copy out of date files and/or directories. Each source file is copied to each host in the destination list. Directories are recursively copied in the same way. *Opt_dest_name* is an optional parameter to rename files. If no **install** command appears in the command list or the destination name is not specified, the source file name is used. Directories in the path name will be created if they do not exist on the remote host. The **–o** *distopts* option as specified above under "OPTIONS", has the same semantics as on the command line except they only apply to the files in the source list. The user name used on the destination host is the same as the local host unless the destination name is of the format ''user@host".

The **notify** command is used to mail the list of files updated (and any errors that may have occurred) to the listed names. If no '@' appears in the name, the destination host is appended to the name (e.g., name1@host, name2@host, ...). In this pre-release version, notification is only delivered to the local host.

The **except** command is used to update all of the files in the source list *except* for the files listed in *name list*. This is usually used to copy everything in a directory except certain files.

The **except_pat** command is like the **except** command except that *pattern list* is a list of regular expressions (see *regex* for details). If one of the patterns matches some string within a file name, that file will be ignored. Note that since '\' is a quote character, it must be doubled to become part of the regular expression. Variables are expanded in *pattern list* but not shell file pattern matching characters. To include a '$', it must be escaped with '\'.

The **special** command is used to specify *sh* commands that are to be executed on the remote host after the file in *name list* is updated or installed. If the *name list* is omitted then the shell commands will be executed for every file updated or installed. *String* starts and ends with '"' and can cross multiple lines in *distfile*. Multiple commands to the shell should be separated by ';'. Commands are executed in the user's home directory on the host being updated. The **special** command can be used to rebuild private databases, etc. after a program has been updated. The following environment variables are set for each **special** command:

FILE The full pathname of the local file that was just updated.

REMFILE
The full pathname of the remote file that was just updated.

BASEFILE
The basename of the remote file that was just updated.

The *cmdspecial* command is similar to the **special** command, except it is executed only when the entire command is completed instead of after each file is updated. The list of files is placed in the environment variable *FILES*. Each file name in *FILES* is separated by a ':' (colon).

If a hostname ends in a ''+'' (plus sign), then the plus is stripped off and NFS checks are disabled. This is equivalent to disabling the **–o***chknfs* option just for this one host.

The following is a small example.

```
HOSTS = ( matisse root@arpa)

FILES = ( /bin /lib /usr/bin /usr/games
       /usr/include/{*.h,{stand,sys,vax*,pascal,machine}/*.h}
       /usr/lib /usr/man/man? /usr/ucb /usr/local/rdist )

EXLIB = ( Mail.rc aliases aliases.dir aliases.pag crontab dshrc
       sendmail.cf sendmail.fc sendmail.hf sendmail.st uucp vfont )

${FILES} -> ${HOSTS}
       install -oremove,chknfs,numchkgroup ;
       except /usr/lib/${EXLIB} ;
       except /usr/games/lib ;
       special /usr/lib/sendmail "/usr/lib/sendmail -bz" ;

srcs:
/usr/src/bin -> arpa
       install -onumchkgroup ;
       except_pat ( \\.o\$ /SCCS\$ ) ;

IMAGEN = (ips dviimp catdvi)

imagen:
/usr/local/${IMAGEN} -> arpa
       install -onumchkgroup /usr/local/lib ;
       notify ralph ;

${FILES} :: stamp.cory
       install -onumchkgroup ;
       notify Administrator ;
```

ENVIRONMENT

Rdist makes use of the following environment variables, if set:

TMPDIR
: Name of temporary directory to use. Default is **$OPENNT_ROOT/tmp**.

RSH
: Name of the default remote shell program to use. Default is **$OPENNT_ROOT/bin/rsh**.

FILES

distfile input command file.

$TMPDIR/rdist*
: temporary file for update lists.

DIAGNOSTICS

??

NOTES

If the basename of a file (the last component in the pathname) is ".", then **rdist** assumes the remote (destination) name is a directory. i.e. **/tmp/.** means that **/tmp** should be a directory on the remote host.

The following options are still recognized for backwards compatibility:

–v –N –O –q –b –r –R –s –w –y –h –i –x

CAVEATS

Source files must reside on the local host where *rdist* is executed.

Group names with spaces in them will cause *rdist* to fail. The *rdist* protocol cannot handle spaces in group names. When the source of the file is on INTERIX, always use the option **numchkgroup**.

Variable expansion only works for name lists; there should be a general macro facility.

Rdist aborts on files which have a negative *mtime* (before Jan 1, 1970).

If a hardlinked file is listed more than once in the same target, then *rdist* will report missing links. Only one instance of a link should be listed in each target.

SEE ALSO

sh, *csh*, *stat*, *rsh*

rcmd()

NAME

rdistd – remote file distribution server program

SYNOPSIS

```
rdistd –S [–D]

rdistd –V
```

DESCRIPTION

Rdistd is the server program for the *rdist* command. It is normally run by *rdist* via *rsh*.

The **–S** argument must be specified. The option is required so that *rdistd* is not accidentally started since it normally resides somewhere in a normal user's *PATH*.

The following options are available:

–D Enable debugging messages. Messages are logged via *syslog*( ).

–V Print version information and exit.

SEE ALSO

rdist, *rsh*

rcmd()

NAME

read – read line of input from standard-input

SYNOPSIS

`read [-prs] [-u[n]] parameter ...`

DESCRIPTION

The *read* utility reads a line of input from standard input, separates the line into fields using the shell's *IFS* parameter, and assigns each field to the specified parameters. If there are more parameters than fields, the extra parameters are set to null, or alternatively, if there are more fields than parameters, the last parameter is assigned the remaining fields (inclusive of any separating spaces). If no parameters are specified, the *REPLY* parameter is used.

In an interactive shell, if the input line ends in a backslash and the **–r** option was not used, the backslash and newline are stripped and more input is read. If no input is read, *read* exits with a non-zero status.

You can specify a prompt which is printed to standard error before any input is read by appending a question mark and the prompt to the first parameter (e.g., **read nfoo?'number of foos: '**).

The *read* command takes the following options:

–p Read from the current co-process.

–r Do not treat the backslash (\) as a special character.

–s Save input to the history file.

–u *n* Read from file descriptor *n*. If no descriptor is given, file descriptor 0 is used.

SEE ALSO

sh

NAME

renice – change priority of a running process

SYNOPSIS

```
renice [-n increment] [-g|-p|-u] ID ...

renice nice_value -p] pid [-g gid ...]
       [-p pid ...] [-u user]
renice nice_value[-g gid] [-p pid ...] [-u user]

renice nice_value[-u user] [-p pid ...] [-u user]
```

DESCRIPTION

The *renice* command changes the priority of running processes; the processes can be specified by process ID, group ID, or user. The *renice* command can only be run by a user with the appropriate privileges, and there is no way to get the appropriate privileges on Windows NT.

The first syntax given is the standard version; the other synopses reflect obsolescent versions. The change in priority is given by *increment*; if you don't give the **–n** option, the priority is not changed.

In the obsolescent versions, the priority of the processes is changed to *nice_value*, which is an absolute priority, not a change in priority.

Renice takes the following options:

–g Treat all operands as process group IDs.

–n *increment* Adjust the scheduling priority of the command by *increment*, a decimal integer. A positive value for *increment* lowers the priority while a negative value increases the priority.

–p Treat all operands as process IDs. This is the default if no options are specified.

–u Treat all operands as users, either as a user name (if a user exists with that name) or as a user ID.

DIAGNOSTICS

On success, *renice* exits with status 0; if an error occurred, it exits with status >0.

NOTE

There is currently no way to get the appropriate privileges to change the scheduling priority of a process.

SEE ALSO

nice

NAME

resize – utility to set TERMCAP and terminal settings to current window size

SYNOPSIS

resize **[**–u**] [–s [***row col***]]**

DESCRIPTION

Resize prints a shell command for setting the *TERM* and *TERMCAP* environment variables to indicate the current size of *xterm* window from which the command is run. For this output to take effect, *resize* must either be evaluated as part of the command line (usually done with a shell alias or function) or else redirected to a file which can then be read in. From the C shell (usually known as **/bin/csh**), the following alias could be defined in the user's **.cshrc**:

```
% alias rs 'set noglob; 'eval resize''
```

After resizing the window, the user would type:

```
% rs
```

Users of versions of the Bourne shell (usually known as **/bin/sh**) that don't have command functions will need to send the output to a temporary file and the read it back in with the ''.'' command:

```
$ resize >/tmp/out
$ . /tmp/out
```

OPTIONS

The following options may be used with *resize*:

–u This option indicates that Bourne shell commands should be generated even if the user's current shell isn't **/bin/sh**.

–c This option indicates that C shell commands should be generated even if the user's current shell isn't */bin/csh*.

–s [*rows columns*]
This option indicates that that Sun console escape sequences will be used instead of the special *xterm* escape code. If *rows* and *columns* are given, *resize* will ask the *xterm* to resize itself. However, the window manager may choose to disallow the change.

FILES

$OPENNT_ROOT/etc/termcap
For the base termcap entry to modify.

~/.cshrc user's alias for the command.

SEE ALSO

csh, *tset*, *xterm*

AUTHORS

Mark Vandevoorde (MIT-Athena), Edward Moy (Berkeley)

Copyright (c) 1984, 1985 by Massachusetts Institute of Technology.
See *X*(1) for a complete copyright notice.

BUGS

The **–u** or **–c** must appear to the left of **–s** if both are specified.

There should be some global notion of display size; termcap and terminfo need to be rethought in the context of window systems. (Fixed in 4.3BSD, and Ultrix-32 1.2)

NAME

rexecd – remote execution server

SYNOPSIS

`rexecd`

DESCRIPTION

Rexecd is the server for the routine. The server provides remote execution facilities with authentication based on user names and passwords.

Rexecd listens for service requests at the port indicated in the ‘‘exec’’ service specification. When a service request is received the following protocol is initiated:

1. The server reads characters from the socket up to a NUL (\0) byte. The resultant string is interpreted as an ASCII number, base 10.
2. If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the *stderr*. A second connection is then created to the specified port on the client’s machine.
3. A NUL terminated user name of at most 128 characters is retrieved on the initial socket (the value is 16 characters for non-INTERIX systems).
4. A NUL terminated, unencrypted password of at most 128 characters is retrieved on the initial socket (the value 16 characters for non-INTERIX systems).
5. A NUL terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system’s argument list.
6. *Rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user’s home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.
7. A NUL byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

username too long
: The name is longer than 128 characters.

password too long
: The password is longer than 128 characters.

command too long
: The command line passed exceeds the size of the argument list (as configured into the system).

Login incorrect.
: No user database entry for the user name existed or the wrong

password was supplied.

No remote directory.
The *chdir* command to the home directory failed.

Try again.
A *fork* by the server failed.

<shellname>: ...
The user's login shell could not be started. This message is returned on the connection associated with the *stderr*, and is not preceded by a flag byte.

SEE ALSO

rexec

NAME

rlog – print log messages and other information about RCS files

SYNOPSIS

```
rlog [-bhLrt] [-d date] [-l lockers]
     [-r revisions] [-s states] [-V n]
     [-w logins] [-x suffixes] file ...
```

DESCRIPTION

The *rlog* utility prints information about RCS files.

Pathnames matching an RCS suffix denote RCS files; all others denote working files. Names are paired as explained in *ci*.

rlog prints the following information for each RCS file: RCS pathname, working pathname, head (i.e., the number of the latest revision on the trunk), default branch, access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, *rlog* prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. All times are displayed in Coordinated Universal Time (UTC). Without options, *rlog* prints complete information. The options below restrict this output.

–b Print information about the revisions on the default branch, normally the highest branch on the trunk.

–d*dates* Print information about revisions with a checkin date/time in the ranges given by the semicolon-separated list of *dates*. A range of the form *d1*<*d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2* inclusive. A range of the form <*d* or *d*> selects all revisions dated *d* or earlier. A range of the form *d*< or >*d* selects all revisions dated *d* or later. A range of the form *d* selects the single, latest revision dated *d* or earlier. The date/time strings *d*, *d1*, and *d2* are in the free format explained in *co*. Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.

–h Print only the RCS pathname, working pathname, head, default branch, access list, locks, symbolic names, and suffix.

–L Ignore RCS files that have no locks set. This is convenient in combination with **–h**, **–l**, and **–R**.

–l*lockers* Print information about locked revisions only. In addition, if the comma-separated list *lockers* of login names is given, ignore all locks other than those held by the *lockers*. For example, **rlog –L –R –lwft RCS/*** prints the name of RCS files locked by the user **wft**.

–R Print only the name of the RCS file. This is convenient for translating a working pathname into an RCS pathname.

–r*revisions*
Print information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1*:*rev2* means revisions *rev1* to *rev2* on the same branch, :*rev* means revisions from the beginning of the branch up to and including *rev*, and *rev*: means revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range. A branch followed by a **.** means the latest revision in that branch. A bare **–r** with no *revisions* means the latest revision on the default branch, normally the trunk.

–s*states* Print information about revisions whose state attributes match one of the states given in the comma-separated list *states*.

–t Print the same as **–h**, plus the descriptive text.

–V*n* Emulate RCS version *n* when generating logs. See *co* for more.

–w*logins* Print information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

–x*suffixes* Use *suffixes* to characterize RCS files. See *ci* for details.

rlog prints the intersection of the revisions selected with the options **–d**, **–l**, **–s**, and **–w**, intersected with the union of the revisions selected by **–b** and **–r**.

EXAMPLES

This command prints the names of all RCS files in the subdirectory **RCS** that have locks:

```
$ rlog  -L  -R  RCS/*
```

Print the headers of those files:

```
$ rlog  -L  -h  RCS/*
```

Print the headers plus the log messages of the locked revisions:

```
$ rlog  -L  -l  RCS/*
```

Print complete information about all RCS files in the subdirectory **RCS**:

```
$ rlog  RCS/*
```

ENVIRONMENT

RCSINIT
A list of options prepended to the argument list, separated by spaces. See *ci* for details.

DIAGNOSTICS

The exit status is zero if and only if all operations were successful.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 1.8; Release Date: 1997 / 11 / 12.
Copyright © 1982, 1988, 1989 by Walter F. Tichy.
Copyright © 1990, 1991 by Paul Eggert.

BUGS

The separator for revision ranges in the **–r** option used to be **–** instead of **:**, but this leads to confusion when symbolic names contain **–**. For backwards compatibility **rlog –r** still supports the old **–** separator, but it warns about this obsolete use.

SEE ALSO

ci, *co*, *ident*, *rcs*, *rcsdiff*, *rcsmerge*
Walter F. Tichy, RCS—A System for Version Control, *Software—Practice & Experience* **15**, 7 (July 1985), 637-654.

NAME

rlogin – remote login

SYNOPSIS

rlogin **[**–8ELDd**]** **[**–e *char***]** **[**–l *username***]** *host*

DESCRIPTION

Rlogin starts a terminal session on a remote host *host*.

Rlogin first attempts to use the Kerberos authorization mechanism, described below. If the remote host does not support Kerberos the standard Berkeley **.rhosts** authorization mechanism is used. The options are as follows:

–8 The **–8** option allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than ^S/^Q.

–D The **–D** option prefixes both the remote and local usernames with the user's Windows NT domainname. (By default, both remote and local usernames are the username without the domain.) The argument to the **–l** option will always override the remote username value. This option should only be used when connecting with another INTERIX system.

–d The **–d** option turns on socket debugging (see *setsockopt*( )) on the TCP sockets used for communication with the remote host.

–E The **–E** option stops any character from being recognized as an escape character. When used with the **–8** option, this provides a completely transparent connection.

–e The **–e** option allows user specification of the escape character, which is ''~'' by default. This specification may be as a literal character, or as an octal value in the form \nnn.

–K The **–K** option turns off all Kerberos authentication.

–L The **–L** option allows the rlogin session to be run in ''litout'' (see *tty*) mode.

–l The **–l** option specifies a different *username* for the remote login. If this option is not specified, your local username will be used.

A line of the form ''<escape char>.'' disconnects from the remote host. Similarly, the line ''<escape char>^Z'' will suspend the *rlogin* session, and ''<escape char><delayed-suspend char>'' suspends the send portion of the rlogin, but allows output from the remote system. By default, the tilde (''~'') character is the escape character, and normally control-Y (''^Y'') is the delayed-suspend character.

All echoing takes place at the remote site, so that (except for delays) the *rlogin* is transparent. Flow control via ^S/^Q and flushing of input and output on interrupts are handled properly.

INTERIX Authentication

On traditional systems, a user can *rlogin* from a system identified in his or her **.rhosts** file without providing a password. This is because on a traditional

system, the *rlogind* program is run as the superuser, and can take any login identity. On INTERIX, this is not possible. Instead, a second file (**$HOME/.password**) is needed which contains the user's unencrypted password. The **.password** file should be created with the *regpwd* utility so that its permissions are correctly set. If the permissions on the file are not correct, the file is ignored.

FILES

The *rcp* and *rshd* programs make use of the following files, if they exist:

$OPENNT_ROOT/etc/hosts.equiv
: On the target system, this file contains the names of systems and users allowed to login. See *rlogind* for more information.

$HOME/.password
: On the target system, the user's password.

$HOME/.rhosts
: On the target system, this file may contain the names of systems (and user names) allowed to login. See *rlogind* for more information.

ENVIRONMENT

The following environment variable is utilized by *rlogin:*

TERM Determines the user's terminal type.

BUGS

More of the environment should be propagated.

SEE ALSO

rlogind, *rsh*

NAME

rlogind – remote login server

SYNOPSIS

`rlogind [-alnQ]`

DESCRIPTION

Rlogind is the server for the *rlogin* program. The server provides a remote login facility with authentication based on privileged port numbers from trusted hosts.

Options supported by *rlogind:*

–a Ask hostname for verification.

–l Prevent any authentication based on the user's **.rhosts** file.

–n Disable keep-alive messages.

–Q Sets "really quiet" mode. Really quiet mode suppresses the entire login banner.

Rlogind listens for service requests at the port indicated in the ''login'' service specification. When a service request is received the following protocol is initiated:

1. The server checks the client's source port. If the port is not in the range 512-1023, the server aborts the connection.
2. The server checks the client's source address and requests the corresponding host name (see *gethostbyaddr*( )). If the hostname cannot be determined, the dot-notation representation of the host address is used. If the hostname is in the same domain as the server (according to the last two components of the domain name), or if the **–a** option is given, the addresses for the hostname are requested, verifying that the name and address correspond. Normal authentication is bypassed if the address verification fails.

Once the source port and address have been checked, *rlogind* proceeds with the authentication process described in *rshd*. It then allocates a pseudo terminal (see *pty*), and manipulates file descriptors so that the slave half of the pseudo terminal becomes the *stdin*, *stdout*, and *stderr* for a login process. The login process is an instance of the *login* program, invoked with the **–f** option if authentication has succeeded. If automatic authentication fails, the user is prompted to log in as if on a standard terminal line.

The parent of the login process manipulates the master side of the pseudo terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. In normal operation, the packet protocol described in *pty* is invoked to provide ^S/^Q type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, *TERM*. The screen or window size of the terminal is requested from the client, and window size changes from the client are propagated to the pseudo terminal.

Transport-level keepalive messages are enabled unless the **–n** option is present. The use of keepalive messages allows sessions to be timed out if the client crashes or becomes unreachable.

INTERIX Authentication

On traditional systems, a user can *rlogin* from a system identified in his or her **.rhosts** file without providing a password. This is because on a traditional system, the *rlogind* program is run as the superuser, and can take any login identity. On INTERIX, this is not possible. Instead, a second file (**$HOME/.password**) is needed which contains the user's unencrypted password. The **.password** file should be created with the *regpwd* utility so that its permissions are correctly set. If the permissions on the file are not correct, the file is ignored.

DIAGNOSTICS

All initial diagnostic messages are indicated by a leading byte with a value of 1, after which any network connections are closed. If there are no errors before *login* is invoked, a null byte is returned as in indication of success.

Try again.
A *fork* by the server failed.

FILES

The *rlogin* and *rlogind* programs make use of the following files, if they exist:

$OPENNT_ROOT/etc/hosts.equiv
On the target system, this file contains the names of systems and users allowed to login. See *rlogind* for more information.

$HOME/.password
On the target system, the user's password.

$HOME/.rhosts
On the target system, this file may contain the names of systems (and user names) allowed to login.

SEE ALSO

login, *rshd*, *iruserok*()

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an ''open'' environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol should be used.

NAME

rm – remove directory entries

SYNOPSIS

`rm [-fiRr] file ...`

DESCRIPTION

The *rm* utility attempts to remove the non-directory type files specified on the command line. If the permissions of the file do not permit writing, and the standard input device is a terminal, the user is prompted (on the standard error output) for confirmation.

The options are as follows:

–f Attempt to remove the files without prompting for confirmation, regardless of the file's permissions. If the file does not exist, do not display a diagnostic message or modify the exit status to reflect an error. The **–f** option overrides any previous **–i** options.

–i Request confirmation before attempting to remove each file, regardless of the file's permissions, or whether or not the standard input device is a terminal. The **–i** option overrides any previous **–f** options.

–R Attempt to remove the file hierarchy rooted in each file argument. If the **–i** option is specified, the user is prompted for confirmation before each directory's contents are processed (as well as before the attempt is made to remove the directory). If the user does not respond affirmatively, the file hierarchy rooted in that directory is skipped.

–r Equivalent to **–R**.

It is an error to attempt to remove the files "." and "..".

DIAGNOSTICS

The *rm* utility exits 0 if all of the named files or file hierarchies were removed, or if the **–f** option was specified and all of the existing files or file hierarchies were removed. If an error occurs, *rm* exits with a value >0.

SEE ALSO

rmdir

NAME

rmdir – remove directories

SYNOPSIS

`rmdir` *`directory ...`*

DESCRIPTION

The *rmdir* utility removes the directory entry specified by each *directory* argument, provided it is empty.

Arguments are processed in the order given. In order to remove both a parent directory and a subdirectory of that parent, the subdirectory must be specified first so the parent directory is empty when *rmdir* tries to remove it.

DIAGNOSTICS

The *rmdir* utility exits with one of the following values:

0 Each directory entry specified by a dir operand referred to an empty directory and was removed successfully.

>0 An error occurred.

SEE ALSO

rm

NAME

rsh – remote shell

SYNOPSIS

`rsh [-Ddn] [-l username] host [command]`

DESCRIPTION

Rsh executes *command* on *host*.

Rsh copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *rsh* normally terminates when the remote command does. The options are as follows:

–D The **–D** option prefixes both the remote and local usernames with the user's Windows NT domainname. (By default, both remote and local usernames are the username without the domain.) The argument to the **–l** option will always override the remote username value. This option should only be used when connecting with another INTERIX system.

–d The **–d** option turns on socket debugging (using *setsockopt*( )) on the TCP sockets used for communication with the remote host.

–l By default, the remote username is the same as the local username. The **–l** option allows the remote name to be specified. Kerberos authentication is used, and authorization is determined as in *rlogin*.

–n The **–n** option redirects input from the special device **/dev/null** (see the "BUGS" section of this manual page).

If no *command* is specified, you will be logged in on the remote host using *rlogin*.

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. For example, the command

```
$ rsh otherhost cat remotefile >> localfile
```

appends the remote file *remotefile* to the local file *localfile,* while

```
rsh otherhost cat remotefile ">>" other_remotefile
```

appends *remotefile* to *other_remotefile.*

INTERIX Authentication

On traditional systems, a user can *rlogin* to a system identified in his or her **.rhosts** file without providing a password. This is because on a traditional system, the *rlogind* program is run as the superuser, and can take any login identity. On INTERIX, this is not possible. Instead, a second file (**$HOME/.password**) is needed which contains the user's unencrypted password. Use the *regpwd* utility to create the **.password** file and set the correct permissions on that file. If the permissions on the file are not correct, the file is ignored.

FILES

%WINDIR%/system32/drivers/etc/hosts
The list of hosts.

BUGS

If you are using *csh* and put a *rsh* in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired you should redirect the input of *rsh* to **/dev/null** using the **–n** option.

You cannot run an interactive command (like *rogue* or *vi*) using *rsh;* use *rlogin* instead.

Stop signals stop the local *rsh* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

SEE ALSO

rlogin

rcmd()

NAME

rshd – remote shell server

SYNOPSIS

`rshd [-alnL]`

DESCRIPTION

The *rshd* server is the server for the *rcmd*( ) routine and, consequently, for the *rsh* program. The server provides remote execution facilities with authentication based on privileged port numbers from trusted hosts.

Options supported by *rshd:*

–a Ask hostname for verification.

–l Prevent any authentication based on the user's **.rhosts** file.

–n Disable keep-alive messages.

–L Log successful accesses very verbosely.

The *rshd* server listens for service requests at the port indicated in the ''cmd'' service specification. When a service request is received the following protocol is initiated:

1. The server checks the client's source port. If the port is not in the range 512-1023, the server aborts the connection.
2. The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.
3. If the number received in step 2 is non-zero, it is interpreted as the port number of a secondary stream to be used for the *stderr*. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 512-1023.
4. The server checks the client's source address and requests the corresponding host name (see *gethostbyaddr*( )). If the hostname cannot be determined, the dot-notation representation of the host address is used. If the hostname is in the same domain as the server (according to the last two components of the domain name), or if the **–a** option is given, the addresses for the hostname are requested, verifying that the name and address correspond. If address verification fails, the connection is aborted with the message, ''Host address mismatch.''
5. A null terminated user name of at most 128 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the *client*'s machine.
6. A null terminated user name of at most 128 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the *server*'s machine.
7. A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8. *Rshd* then validates the user using *iruserok*( ), which uses the file **$OPENNT_ROOT/etc/hosts.equiv**, the **.rhosts** file found in the user's home directory, and the **.password** file also found in the user's home directory. The **–l** option prevents *ruserok*( ) from doing any validation based on the user's **.rhosts** file, unless the user is the superuser.
9. The file **$OPENNT_ROOT/etc/nologin**exists.
10. A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd*.

Transport-level keepalive messages are enabled unless the **–n** option is present. The use of keepalive messages allows sessions to be timed out if the client crashes or becomes unreachable.

The **–L** option causes all successful accesses to be logged to *syslogd* as messages.

INTERIX Authentication

On traditional systems, a user can *rsh* from a system identified in his or her **.rhosts** file without providing a password. This is because on a traditional system, the *rshd* program is run as the superuser, and can take any login identity. On INTERIX, this is not possible. Instead, a second file (**$HOME/.password**) is needed which contains the user's unencrypted password. The files should be mode 600 (see *chmod*), so only the user can read it. Use the *regpwd* utility to create the **.password** file and to ensure that the permissions are correctly set. If the permissions on the file are not correct, the file is ignored.

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 10 above upon successful completion of all the steps prior to the execution of the login shell).

Locuser too long.
The name of the user on the client's machine is longer than 128 characters.

Ruser too long.
The name of the user on the remote machine is longer than 128 characters.

Command too long.
The command line passed exceeds the size of the argument list (as configured into the system).

Remote directory.
The *chdir* command to the home directory failed.

Permission denied.
The authentication procedure described above failed or there is no password file entry for the specified user.

Can't make pipe.
The pipe needed for the *stderr*, wasn't created.

Can't fork; try again.
A *fork* by the server failed.

<shellname>: ...
The user's login shell could not be started. This message is returned on the connection associated with the *stderr*, and is not preceded by a flag byte.

FILES

The *rsh* and *rshd* programs make use of the following files, if they exist:

$OPENNT_ROOT/etc/hosts.equiv
On the target system, this file contains the names of systems and users allowed to login. See *rlogind* for more information.

$HOME/.password
On the target system, the user's password.

$HOME/.rhosts
On the target system, this file may contain the names of systems (and user names) allowed to login.

SEE ALSO

rsh, *rcmd*(), *iruserok*()

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an ''open'' environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol (such as Telnet) should be used.

NAME

s2p – Sed to Perl translator

SYNOPSIS

`s2p [-np] [-Dnumber] filename`

DESCRIPTION

S2p takes a sed script specified on the command line (or from standard input) and produces a comparable *perl* script on the standard output.

Options

Options include:

–D *number*
sets debugging flags.

–n specifies that this sed script was always invoked with a sed -n. Otherwise a switch parser is prepended to the front of the script.

–p specifies that this sed script was never invoked with a sed -n. Otherwise a switch parser is prepended to the front of the script.

Considerations

The *perl* script produced looks very sed-ish, and there may very well be better ways to express what you want to do in perl. For instance, s2p does not make any use of the split operator, but you might want to.

The *perl* script you end up with may be either faster or slower than the original *sed* script. If you're only interested in speed you'll just have to try it both ways. Of course, if you want to do something sed doesn't do, you have no choice. It's often possible to speed up the *perl* script by various methods, such as deleting all references to $\ and chop.

ENVIRONMENT VARIABLES

S2p uses no environment variables.

AUTHOR

Larry Wall <lwall@jpl-devvax.Jpl.Nasa.Gov>

FILES

SEE

perl (the *perl* compiler / interpreter), *a2p* (*awk* to *perl* translator)

DIAGNOSTICS

BUGS

NAME

script – make typescript of terminal session

SYNOPSIS

`script [-a] [file]`

DESCRIPTION

Script makes a typescript of everything printed on your terminal. It is useful for students who need a hardcopy record of an interactive session as proof of an assignment, as the typescript file can be printed out later with *lp*.

If the argument *file* is given, *script* saves all dialogue in *file*. If no file name is given, the typescript is saved in the file **typescript**.

Option:

–a Append the output to *file* or **typescript**, retaining the prior contents.

The script ends when the forked shell exits (to exit *sh* on INTERIX systems, use *control-D*; to exit *csh*, use *exit, logout* or, if *ignoreeof* is not set, *control-d*).

Certain interactive commands, such as *vi*, create garbage in the typescript file. *Script* works best with commands that do not manipulate the screen, the results are meant to emulate a hardcopy terminal.

ENVIRONMENT

The following environment variable is utilized by *script:*

SHELL If the variable *SHELL* exists, the shell forked by *script* will be that shell. If *SHELL* is not set, the Bourne shell is assumed. (Most shells set this variable automatically).

NOTES

Script places *everything* in the log file, including linefeeds and backspaces. This is not what the naive user expects.

SEE ALSO

csh (for the *history* mechanism).

NAME

sed – stream editor

SYNOPSIS

```
sed [-an] command [file ...]
sed [-an] [-e command] [-f command_file] [file ...]
```

DESCRIPTION

The *sed* utility reads the specified files, or the standard input if no files are specified, modifying the input as specified by a list of commands. The input is then written to the standard output.

A single command may be specified as the first argument to *sed*. Multiple commands may be specified by using the **–e** or **–f** options. All commands are applied to the input in the order they are specified regardless of their origin.

The following options are available:

–a The files listed as parameters for the **w** functions are created (or truncated) before any processing begins, by default. The **–a** option causes *sed* to delay opening each file until a command containing the related **w** function is applied to a line of input.

–e *command*
Append the editing commands specified by the *command* argument to the list of commands.

–f *command_file*
Append the editing commands found in the file *command_file* to the list of commands. The editing commands should each be listed on a separate line.

–n By default, each line of input is echoed to the standard output after all of the commands have been applied to it. The **–n** option suppresses this behavior.

The form of a *sed* command is as follows:

```
[address[,address]]function[arguments]
```

Whitespace may be inserted before the first address and the function portions of the command.

Normally, *sed* cyclically copies a line of input, not including its terminating newline character, into a *pattern space*, (unless there is something left after a **D** function), applies all of the commands with addresses that select that pattern space, copies the pattern space to the standard output, appending a newline, and deletes the pattern space.

Some of the functions use a *hold space* to save all or part of the pattern space for subsequent retrieval.

Sed Addresses

sed commands do not require an address, but if one is specified, it must be one of

- a number that counts input lines cumulatively across input fields;
- a dollar character ("$") that addresses the last line of input; or
- a context address which consists of a regular expression surrounded by delimiter characters.

A command line with no addresses selects every pattern space.

A command line with one address selects all of the pattern spaces that match the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only that line is selected.) Starting at the first line following the selected range, *sed* starts looking again for the first address.

Editing commands can be applied to non-selected pattern spaces by use of the exclamation character ("!") function.

Sed Regular Expressions

The *sed* regular expressions are basic regular expressions (BRE). *sed also* has the following two additions to BRE's:

1. In a context address, any character other than a backslash ("\") or newline character may be used to delimit the regular expression. To do this, precede the first delimiting character with a backslash; for example, to make x the delimiting character for the BRE "//*", you would type "\x//*x". The second x is automatically treated as the closing delimiter. (This is often done to make it easier to type regular expressions containing slashes.) Putting a backslash character before the delimiting character causes the character to be treated literally. For example, in the context address \xabc\xdefx, the RE delimiter is an x and the second x stands for itself, so that the regular expression is "abcxdef".
2. The escape sequence \n matches a newline character embedded in the pattern space. You can't, however, use a literal newline character in an address or in the substitute command.

One special feature of *sed* regular expressions is that they can default to the last regular expression used. If a regular expression is empty, i.e. just the delimiter characters are specified, the last regular expression encountered is used instead. The last regular expression is defined as the last regular expression used as part of an address or substitute command, and at run-time, not compile-time. For example, the command "/abc/s//XXX/" will substitute "XXX" for the pattern "abc".

Sed Functions

In the following list of commands, the maximum number of permissible addresses for each command is indicated by [0addr], [1addr], or [2addr], representing zero, one, or two addresses.

The argument *text* consists of one or more lines. To embed a newline in the text, precede it with a backslash. Other backslashes in text are deleted and the following character taken literally.

The **r** and **w** functions take an optional file parameter, which should be separated from the function letter by white space. Each file given as an argument to *sed* is created (or its contents truncated) before any input processing begins.

The **b**, **r**, **s**, **t**, **w**, **y**, **!**, and **:** functions all accept additional arguments. The following synopses indicate which arguments have to be separated from the function letters by white space characters.

Two of the functions take a function-list. This is a list of *sed* functions separated by newlines, as follows:

```
{ function
  function
 ...
  function
}
```

The **{** can be preceded by white space and can be followed by white space. The function can be preceded by white space. The terminating **}** must be preceded by a newline or optional white space.

[*2addr*] *function-list*
: Execute function-list only when the pattern space is selected.

[*1addr*]**a**
text
: Write *text* to standard output immediately before each attempt to read a line of input, whether by executing the **N** function or by beginning a new cycle.

[*2addr*]**b**[*label*]
: Branch to the **:** function with the specified label. If the label is not specified, branch to the end of the script.

[*2addr*]**c**
text
: Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, *text* is written to the standard output.

[*2addr*]**d**
: Delete the pattern space and start the next cycle.

[*2addr*]**D**
: Delete the initial segment of the pattern space through the first newline character and start the next cycle.

[*2addr*]**g**
: Replace the contents of the pattern space with the contents of the hold space.

[*2addr*]**G**
: Append a newline character followed by the contents of the hold space to the pattern space.

[*2addr*]**h**
: Replace the contents of the hold space with the contents of the pattern space.

[*2addr*]**H**

Append a newline character followed by the contents of the pattern space to the hold space.

[*1addr*]**i**
text

Write *text* to the standard output.

[*2addr*]**l**

(The letter ell.) Write the pattern space to the standard output in a visually unambiguous form. This form is as follows:

backslash	\
alert	\a
form-feed	\f
newline	\n
carriage-return	\r
tab	\t
vertical tab	\v

Nonprintable characters are written as three-digit octal numbers (with a preceding backslash) for each byte in the character (most significant byte first). Long lines are folded, with the point of folding indicated by displaying a backslash followed by a newline. The end of each line is marked with a "$".

[*2addr*]**n**

Write the pattern space to the standard output if the default output has not been suppressed, and replace the pattern space with the next line of input.

[*2addr*]**N**

Append the next line of input to the pattern space, using an embedded newline character to separate the appended material from the original contents. Note that the current line number changes.

[*2addr*]**p**

Write the pattern space to standard output.

[*2addr*]**P**

Write the pattern space, up to the first newline character to the standard output.

[*1addr*]**q**

Branch to the end of the script and quit without starting a new cycle.

[*1addr*]**r** *file*

Copy the contents of *file* to the standard output immediately before the next attempt to read a line of input. If *file* cannot be read for any reason, it is silently ignored and no error condition is set.

[*2addr*]**s**/*regular expression*/*replacement*/*flags*

Substitute the replacement string for the first instance of the regular expression in the pattern space. Any character other than backslash or newline can be used instead of a slash to delimit the RE and the replacement. Within the RE and the replacement, the RE delimiter

itself can be used as a literal character if it is preceded by a backslash.

An ampersand ("&") appearing in the replacement is replaced by the string matching the RE. The special meaning of "&" in this context can be suppressed by preceding it by a backslash. The string "*d*", where *d* is a digit, is replaced by the text matched by the corresponding backreference expression.

A line can be split by substituting a newline character into it. To specify a newline character in the replacement string, precede it with a backslash.

The value of *flags* in the substitute function is zero or more of the following:

0 ... 9 — Make the substitution only for the N'th occurrence of the regular expression in the pattern space.

g — Make the substitution for all non-overlapping matches of the regular expression, not just the first one.

p — Write the pattern space to standard output if a replacement was made. If the replacement string is identical to that which it replaces, it is still considered to have been a replacement.

w *file* — Append the pattern space to *file* if a replacement was made. If the replacement string is identical to that which it replaces, it is still considered to have been a replacement.

[*2addr*]**t** [*label*]
Branch to the **:** function bearing the label if any substitutions have been made since the most recent reading of an input line or execution of a **t** function. If no label is specified, branch to the end of the script.

[*2addr*]**w** *file*
Append the pattern space to the *file*.

[*2addr*]**x**
Swap the contents of the pattern and hold spaces.

[*2addr*]**y/***string1/string2/*
Replace all occurrences of characters in *string1* in the pattern space with the corresponding characters from *string2*. Any character other than a backslash or newline can be used instead of a slash to delimit the strings. Within *string1* and *string2*, a backslash followed by any character other than a newline is that literal character, and a backslash followed by an "n" is replaced by a newline character.

[*2addr*]**!***function*
[*2addr*]**!***function-list*
Apply the function or function-list only to the lines that are *not* selected by the address(es).

[*0addr*]**:***label*
This function does nothing; it bears a label to which the **b** and **t** commands may branch.

[*1addr*]**=**

Write the line number to the standard output followed by a newline character.

[*0addr*]

Empty lines are ignored.

[*0addr*]**#**

The **#** and the remainder of the line are ignored (treated as a comment), *except* if the first two characters in the file are "#n", then the default output is suppressed. (This is the same as specifying the **–n** option on the command line.)

The *sed* utility exits 0 on success and >0 if an error occurs.

EXAMPLES

To change all backslashes in to forward slashes:

```
s;\\;/;g
```

To delete the first blank line following a line all in uppercase:

```
/^[^[:lower:]][^[:lower:]]*/{
        n
        /^$/d
}
```

(Note the use of the POSIX regular character class [:lower:].)

Change all of the characters in the line to uppercase:

```
/.*/y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
```

SEE ALSO

awk, *ed*, *grep*

NAME

service – administer a INTERIX / INTERIX service

SYNOPSIS

```
service grant user
service help [subcommand]
service install [-n name] [-d displayname] [-u user]
        [-p password] [-s starttype] pathname [args...]
service list [-a]
service remove name
service revoke user
service start service
service stop service
```

DESCRIPTION

The *service* utility installs and administers INTERIX programs being run as Windows NT services. You need Administrator privileges to run *service* with any subcommands other than **help** or **list**.

Each service has a name, which is unique on the system. The name is the basename of the command pathname, or it can be specified when the service is installed by using the **–n** option.

For more information, and for guidelines on how to write a INTERIX service program, see the INTERIX User's Guide.

Service has the following subcommands.

grant *user*
: Grant the service-logon privilege to the specified *user*. A user must possess the service-logon privilege if a service is be run as that user; otherwise, the service will fail to start. Note that the **install** subcommand grants this privilege automatically if the **-u** option is specified.

install [**-n** *name*] [**-d** *displayname*] [**-u** *user*] [**-p** *password*] [**-s** *starttype*] *pathname* [*args...*]
: Install the program specified by *pathname* as a service, with the specified arguments *args*. The *pathname* must be the absolute pathname of a INTERIX executable residing on a local filesystem. If the program is on a network drive, the install will fail. (The **Windows NT Service Control Manager** runs under a privileged account that does not have access to networked filesystems.) The **install** options are explained below.

list [**-a**]
: List installed INTERIX services. If the **-a** option is specified, then both Win32 and INTERIX services will be listed.

remove *name*
: Remove the specified *service*.

revoke *user*
: Revoke the specified *user*'s service-logon privilege. (If a user name and password are supplied to the install subcommand, then the user will be granted the service-logon privilege. However, the remove

subcommand does not revoke this privilege.)

start *name*
Start the service that has the specified *name.*

stop *name*
Stop (kill) the service with the specified *name.*

When installing a service, the following options are recognized:

–d *displayname*
Display the name *displayname* in the Services applet of the Control Panel. The displayname defaults to the name specified by the **-n** option.

–n *name*
Give the service the specified *name.* The name defaults to the basename of the service program's pathname. Each installed service must have a unique name.

–p *passwd*
The password for the *user.*

–s *starttype*
Start the service as the specified *starttype*, which is one of *auto* (start the service at boot time) or *manual* (start only on request).

–u *user* Run service with the permissions of *user.* If a user is specified, a password must also be supplied. If no user is specified, the service runs as the built-in user "LocalSystem", and no password is required.

EXIT STATUS

The *service* utility returns 0 on success and >0 on failure. It is not a failure if the service does not start.

FILES

The *service* command makes use of the following files:

$SYSTEMROOT/system32/psxrun.exe
The Win32 program that mediates between the INTERIX and Win32 service mechanisms. "SYSTEMROOT" is the installed directory for Windows NT, typically **//C/WINNT/**.

NOTES

As mentioned above, services do not have access to networked filesystems. Services cannot automatically mount drives. A user logged in through a service (such as *telnetd*), has access to any drives mounted by the current user sitting at the physical console, using their mappings (subject to the usual permissions). Since these mappings change from user to user, it is not a good idea to rely on the existence of any networked drives.

It may be tempting to try to mount the drive while logged in through *telnet*, for example, but this makes the drive unavailable to users at the physical console and has unpleasant interactions with the Win32 login program. Do not use *NET.EXE* from a *telnet* session.

SEE ALSO

cronadmin

NAME

sh – command interpreter (shell)

SYNOPSIS

`sh [-/+aCeflnuvxIimsVEb] [-/+o` *`longname`*`] [`*`arg ...`*`]`

DESCRIPTION

Sh is the standard command interpreter for the system. The file **$OPENNT_ROOT/bin/sh** is a copy of the KornShell, *ksh*. For full POSIX conformance, invoke with the option **–o posix**.

The KornShell is described in the *ksh* man page; the effects of setting POSIX mode (and the defaults) are described in the section “POSIX Mode”.

SEE ALSO

ksh

NAME

showfont - font dumper for X font server

SYNOPSIS

`showfont [-options ...] [-fn` *`pattern`*`]`

DESCRIPTION

Showfont displays data about the font that matches the given *pattern*. The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

OPTIONS

–server *host:port*
This option specifies the X font server to contact.

–l This option indicates that the bit order of the font is LSBFirst (least significant bit first).

–m This option indicates that the bit order of the font is MSBFirst (most significant bit first).

–L This option indicates that the byte order of the font is LSBFirst (least significant byte first).

–M This option indicates that the byte order of the font is MSBFirst (most significant byte first).

–extents_only
This option indicates that only the font's extents should be displayed.

–start *#*
This option indicates the start of the range of the characters to display (# is a number).

–end *#* This option indicates the end of the range of the characters to display (# is a number).

–unit *#* This option specifies the scanline unit of the font (1, 2, 4 or 8). (# is a number).

–pad *#*

–unit *#*
This option specifies the scanpad unit of the font (1, 2, 4 or 8). (# is a number).

–bitmap_pad *#*
This option specifies the bitmap padding unit of the font (0, 1, or 2, where 0 is ImageRectMin, 1 is ImageRectMaxWidth and 2 is ImageRectMaxWidth).

ENVIRONMENT VARIABLES

FONTSERVER

to get the default host and port to use.

SEE ALSO

fs, *showfont*, *xlsfonts*

COPYRIGHT

Copyright 1991, Network Computing Devices, Inc
See *X* for a full statement of rights and permissions.

AUTHOR

Dave Lemke, Network Computing Devices, Inc

NAME

size – list section sizes and total size

SYNOPSIS

`size [-A|-B] [-d|-o|-x] [-V] objfile ...`

DESCRIPTION

Size lists the section sizes and the total size for each specified *objfile*. By default, one line of output is generated for each object file or each module in an archive.

The *size* utility accepts the following options:

–A Display output in System V format.

–B Display output in BSD format (the default).

–d Display numbers in decimal format. The total size is given in both decimal and hexadecimal.

–o Display numbers in octal format. The total size is given in both octal and hexadecimal.

–V Display version number information on *size* itself.

–x Display numbers in hexadecimal format. The total size is given in both decimal and hexadecimal.

The calculation of text, data and bss deserves discussion. In BSD-style systems, the text size is the sum of all of the text and read-only sections (**.text** and **.rdata**), the data size is the sum of the non-readonly **.data** sections, and the bss size is the sum of all of the **.bss** sections that are left.

For the INTERIX *size* utility, text size is the sum of all sections which are read-only or which contain code (the **.rdata** and **.text** sections). The data size is the sum of all non-bss read / write sections (the **.data** and **.idata** sections). The bss size is the remainder — all uninitialized data.

DIAGNOSTICS

The *size* utility exits with a status of 0 for success, and >0 for failure.

SEE ALSO

ar

NAME

sleep – suspend execution for an interval of time

SYNOPSIS

```
sleep seconds
```

DESCRIPTION

The *sleep* command suspends execution for a minimum of *seconds*. *Sleep* is used to schedule the execution of other commands (see EXAMPLES below).

DIAGNOSTICS

The *sleep* utility exits with one of the following values:

0 On successful completion, or if the signal `SIGALRM` was received.

>0 An error occurred.

EXAMPLES

To schedule the execution of a command for 1800 seconds later:

```
(sleep 1800; sh command_file >& errors)&
```

This incantation would wait a half hour before running the script "command_file".

NAME

sort – sort or merge text files

SYNOPSIS

```
sort [-cmubdfinr] [-t char] [-T char]
     [-k field1[,field2]] ... [ -o output] [file] ...
```

DESCRIPTION

The *sort* utility sorts text files by lines. Comparisons are based on one or more sort keys extracted from each line of input, and are performed lexicographically. By default, if keys are not given, *sort* regards each input line as a single field.

The following options are available:

–c Check that the single input file is sorted. If the file is not sorted, *sort* produces the appropriate error messages and exits with code 1; otherwise, *sort* returns 0. *Sort* **–c** produces no output.

–m Merge only; the input files are assumed to be pre-sorted.

–o *output*
The argument given is the name of an *output* file to be used instead of the standard output. This file can be the same as one of the input files.

–u Unique: suppress all but one in each set of lines having equal keys. If used with the **–c** option, check that there are no lines with duplicate keys.

The following options override the default ordering rules. When ordering options appear independent of key field specifications, the requested field ordering rules are applied globally to all sort keys. When attached to a specific key (see **–k**), the ordering options override all global ordering options for that key.

–d Only blank space and alphanumeric characters are used in making comparisons.

–f Considers all lowercase characters that have uppercase equivalents to be the same for purposes of comparison.

–i Ignore all non-printable characters.

–n An initial numeric string, consisting of optional blank space, optional minus sign, and zero or more digits (including decimal point) is sorted by arithmetic value. (The **–n** option no longer implies the **–b** option.)

–r Reverse the sense of comparisons.

The treatment of field separators can be altered using the options:

–b Ignores leading blank space when determining the start and end of a restricted sort key. A **–b** option specified before the first **–k** option applies globally to all **–k** options. Otherwise, the **–b** option can be attached independently to each *field* argument of the **–k** option (see below). Note that the **–b** option has no effect unless key fields are specified.

–t *char* *Char* is used as the field separator character. The initial *char* is not considered to be part of a field when determining key offsets (see below). Each occurrence of *char* is significant (for example, *charchar* delimits an empty field). If **–t** is not specified, blank space characters are used as default field separators, and any number of continuous blanks count as a single field separator.

–T *char* *Char* is used as the record separator character. The default record separator is newline (one line is one record). This should be used with discretion; **–T**<alphanumeric> usually produces undesirable results.

–k *field1*[**,***field2*]
Designates the starting position, *field1*, and optional ending position, *field2*, of a key field. Counting starts at 1 The **–k** option replaces the obsolescent options **+***pos1* and **–***pos2*.

You can specify more than one sorting field by specifying multiple **–k** options. The second field specified is only used to sort records where the first fields specified are identical, and so on.

The following operands are available:

File The pathname of a file to be sorted, merged, or checked. If no file operands are specified, or if a file operand is **–** the standard input is used.

Field A field is defined as a minimal sequence of characters followed by a field separator or a newline character. By default, the first blank space of a sequence of blank spaces acts as the field separator. All blank spaces in a sequence of blank spaces are considered as part of the next field; for example, all blank spaces at the beginning of a line are considered to be part of the first field.

Fields are specified by the **–k**field1[,field2] argument. A missing *field2* argument defaults to the end of a line.

The arguments *field1* and *field2* have the form *m*.*n* followed by one or more of the options **b**,**d**,**f**, **n**,**r**. A *field1* position specified by *m.n* *(m,n>0)* is interpreted as the *n*th character in the *m*th field. A missing .*n* in *field1* means .1 , indicating the first character of the *m*th field; If the **–b** option is in effect, *n* is counted from the first non-blank character in the *m*th field; *m.1b* refers to the first non-blank character in the *m*th field.

A *field2* position specified by *m.n* is interpreted as the *n*th character (including separators) of the *m*th field. A missing .*n* indicates the last character of the *m*th field; *m* = 0 designates the end of a line. Thus the option **–k**v.x,w.y is synonymous with the obsolescent option +v-1.x-1 w-1.y ; when *y* is omitted, **–k**v.x,w is synonymous with **+***v***-1.***x***-1w+1.0**. The obsolescent **+***pos1* **–***pos2* option is still supported, except for **w.0b**, which has no **–k** equivalent.

FILES

The *sort* utility makes use of the following files:

$OPENNT_ROOT/tmp/sort.*
Default temporary directory, used if *TMPDIR* is not set.

outputPID
Temporary name for *output* if *output* already exists.

DIAGNOSTICS

Sort exits with one of the following values:

0 normal behavior.

1 on disorder (or non-uniqueness) with the **–c** option

2 an error occurred.

NOTES

The current *sort* command uses lexicographic radix sorting, which requires that sort keys be kept in memory (as opposed to previous versions which used quick and merge sorts and did not). Thus performance depends highly on efficient choice of sort keys, and the **–b** option and the *field2* argument of the **–k** option should be used whenever possible. Similarly, *sort* **–k1f** is equivalent to *sort* **–f** and may take twice as long.

SEE ALSO

comm, *uniq*, *join*

NAME

spell – check spelling in text

SYNOPSIS

`spell [-bvx] [+local_dict] [file ...]`

DESCRIPTION

The *spell* utility compares words from the named *file*s (or standard input if no files are named) with those in a spelling list. It writes to standard output all words which were not found on the list.

Spell takes the following options:

–b Check against a British dictionary. The default dictionary is American English. The British dictionary contains spellings such as “colour”, “behaviour”, and “recognise”.

–v Write all words not in the spelling list. This is the default behavior. Some implementations of *spell* also check against possible permutations of the words in the spelling list; for those systems, this option turns off that behavior.

+*local_dict*

Treat the words in the file *local_dict* as correctly spelled. They will not appear in the *spell* output. These words are checked in addition to the usual dictionary file.

DIAGNOSTICS

The *spell* command exits with a status of 0 on success, and >0 if an error occurred.

NOTE

The *spell* command is a wrapper around the *ispell* command. For information on dictionary files, formats, and other options which may be supported, see the reference page for *ispell*.

FILES

The *spell* command makes use of the following files:

$OPENNT_ROOT/usr/share/spell/british.hash

British English dictionary extensions.

$OPENNT_ROOT/usr/share/spell/english.hash

American english dictionary file (the default).

SEE ALSO

ispell

NAME

split – split a file into pieces

SYNOPSIS

`split [-b byte_count[k|m]] [-l line_count] [file [name]]`

DESCRIPTION

The *split* utility reads the given *file* (or standard input if no file is specified) and breaks it up into files of 1000 lines each.

The options are as follows:

–b Create smaller files *byte_count* bytes in length. If **k** is appended to the number, the file is split into *byte_count* kilobyte pieces. If **m** is appended to the number, the file is split into *byte_count* megabyte pieces.

–l Create smaller files *n* lines in length.

If additional arguments are specified, the first is used as the name of the input file which is to be split. If a second additional argument is specified, it is used as a prefix for the names of the files into which the file is split. In this case, each file into which the file is split is named by the prefix followed by a lexically ordered suffix in the range of “aa-zz”.

If the *name* argument is not specified, the file is split into lexically ordered files named in the range of “xaa-zzz”.

BUGS

For historical reasons, if you specify *name*, *split* can only create 676 separate files. The default naming convention allows 2028 separate files.

NAME

ssimda – deliver mail to local mailbox

SYNOPSIS

```
ssimda address ...
ssimda [-x]
```

DESCRIPTION

The *ssimda* utility is a simple mail delivery agent. It reads an e-mail message from standard input and delivers it to the local mailbox specified by each *address* on the command line. Network addresses are not supported; addresses containing an @ or ! are ignored.

The following options are supported:

–x Print a usage message to standard error.

Ssimda makes no assumptions about message structure: it doesn't examine the message headers. However, for compatibility with most other mail systems, the message on standard input should be in RFC822 format (with the usual exceptions for MIME and so on).

The text on standard input is appended to the mailbox of *address*; each message is preceded by a blank line and begins with the line

```
From username
```

where *username* is the name of the sender. If a line in the body of the message begins with "From " it will be prepended with ">" to prevent confusion with the beginning of a message.

The maximum length of a line in a mail message is {POSIX2_LINE_MAX} bytes; to see the value of this limit on your system, use the *getconf* utility.

DIAGNOSTICS

Ssimda returns the following exit values:

0 Successful completion.

1 An error occurred.

FILES

The *ssimda* utility uses the following files:

$OPENNT_ROOT/usr/spool/mail/

The mail directory; *ssimda* puts local mail in this directory.

SEE ALSO

mailx

NAME

strerror – output error strings

SYNOPSIS

`strerror [-lx] errnumber ...`

DESCRIPTION

The *strerror* utility writes the error string associated with the *errnumber* to standard-out.

The options are as follows:

–l List all known error codes.

–x Write an explanation of the command to standard-error.

DIAGNOSTICS

The *strerror* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

strsignal

NAME

strsignal – write signal identity

SYNOPSIS

`strsignal [–lmwx] signo ...`

DESCRIPTION

The *strsignal* utility writes the textual description of the signal number as specified by the *signo* operand to standard output.

The options are as follows:

–l List all signals to standard-output.

–m Write the mapping of POSIX.2 signal numbers to NT signals.

–w Suppress writing the warning message about signal number mapping on NT when performing a list operation.

–x Write an explanation of the command to standard-error.

INTERIX maps signals the same as most traditional systems, but there are areas where different historical behaviors are incompatible. POSIX.2 mandates that certain historical numbers be recognised by the *kill* command. Issuing a `kill -9` will send a *SIGKILL* to the receiving process, regardless of the actual value of the SIGKILL macro. Use signal names for clarity.

DIAGNOSTICS

The *strsignal* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

kill, *strerror*

NAME

strings – list printable strings in file

SYNOPSIS

`strings [-a] [-t format] [-n number] file ...`

DESCRIPTION

The *strings* utility searches the specified *file* for printable strings and writes them to standard output, one string to a line. A printable string is a sequence of *number* or more characters terminated by a newline or null character; the default value for *number* is 4.

Strings takes the following options:

–a Scan all of the file.

–n *number*
Display only strings that are *number* characters long or longer; the default value for *number* is 4.

–t *format*
Before each string, write its byte offset from the start of the file. Use the specified *format* for numeric values. The *format* is one of **d** (decimal — the default), **o** (octal), or **x** (hexadecimal). The offset is written on the line before the string.

DIAGNOSTICS

Strings returns 0 if it completed successfully, and >0 if an error occurred.

SEE ALSO

nm

NAME

strip – strip extra information from executable

SYNOPSIS

`strip` *`file`* `...`

DESCRIPTION

The *strip* utility removes any unnecessary information from an executable. It's equivalent to the **–s** option to *c89*.

File is an executable file.

DIAGNOSTICS

The *strip* utility returns an exit status of 0 if it was successful, and >0 if an error occurred.

SEE ALSO

ar, *c89*

NAME

stty – set the options for a terminal device interface

SYNOPSIS

`stty [-a | -e | -g] [-f file] [operands]`

DESCRIPTION

The *stty* utility sets or reports on terminal characteristics for the device that is its standard input. If no options or operands are specified, it reports the settings of a subset of characteristics as well as additional ones if they differ from their default values. Otherwise it modifies the terminal state according to the specified arguments (if it is able). Some combinations of arguments are mutually exclusive on some terminal types. INTERIX uses a terminal type of `opennt` to describe the console.

The following options are available:

–a Display all the current settings for the terminal to standard output.

–e Display all the current settings for the terminal to standard output in the traditional BSD ‘‘all’’ and ‘‘everything’’ formats.

–f Open and use the terminal named by *file* rather than using standard input. The file is opened using the O_NONBLOCK flag of *open*( ), making it possible to set or display settings on a terminal that might otherwise block on the open.

–g Display all the current settings for the terminal to standard output in a form that may be used as an argument to a subsequent invocation of *stty* to restore the current terminal state.

The following arguments are available to set the terminal characteristics:

Control Modes

These flags affect hardware characteristics associated with the terminal

Input Modes

These flags affect the characters recognized on input.

Output Modes

These flags affect the characters recognized on output.

Control Characters

These flags define the control characters recognized by the terminal.

Combination Modes

These flags represent useful combinations of other flags.

Compatibility Modes

These flags exist for compatibility with older versions of *stty*.

Control Modes

Control mode flags affect hardware characteristics associated with the terminal. This corresponds to the *c_cflag* in the **termios** structure.

parenb (**-parenb**)
: Enable (disable) parity generation and detection.

parodd (**-parodd**)
: Select odd (even) parity.

cs5 cs6 cs7 cs8
: Select character size, if possible.

number Set terminal baud rate to the number given, if possible. If the baud rate is set to zero, modem control is no longer asserted.

ispeed *number*
: Set terminal input baud rate to the number given, if possible. If the input baud rate is set to zero, the input baud rate is set to the value of the output baud rate.

ospeed *number*
: Set terminal output baud rate to the number given, if possible. If the output baud rate is set to zero, modem control is no longer asserted.

speed *number*
: This sets both **ispeed** and **ospeed** to *number*.

hupcl (**-hupcl**)
: Stop asserting modem control (do not stop asserting modem control) on last close, if possible.

hup (**-hup**)
: Same as hupcl (-hupcl).

cstopb (**-cstopb**)
: Use two (one) stop bits per character.

cread (**-cread**)
: Enable (disable) the receiver.

clocal (**-clocal**)
: Assume a line without (with) modem control.

crtscts (**-crtscts**)
: Enable RTS / CTS flow control.

Input Modes

This corresponds to the *c_iflag* in the **termios** structure.

ignbrk (**-ignbrk**)
: Ignore (do not ignore) break on input.

brkint (**-brkint**)
: Signal (do not signal) INTR on break.

ignpar (**-ignpar**)
: Ignore (do not ignore) parity errors.

parmrk (**-parmrk**)
: Mark (do not mark) parity errors.

inpck (**-inpck**)
: Enable (disable) input parity checking.

istrip (**-istrip**)
: Strip (do not strip) input characters to seven bits.

inlcr (**-inlcr**)

Map (do not map) NL to CR on input.

igncr (**-igncr**)
Ignore (do not ignore) CR on input.

icrnl (**-icrnl**)
Map (do not map) CR to NL on input.

ixon (**-ixon**)
Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START , or if **ixany** is set, any character restarts output.

ixoff (**-ixoff**)
Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

ixany (**-ixany**)
Allow any character (allow only START) to restart output.

imaxbel (**-imaxbel**)
The system imposes a limit of MAX_INPUT (currently 512) characters in the input queue. If *imaxbel* is set and the input queue limit has been reached, subsequent input causes the system to send an ASCII BEL character to the output queue (the terminal beeps at you). Otherwise, if *imaxbel* is unset and the input queue is full, the next input character causes the entire input and output queues to be discarded.

Output Modes

This corresponds to the *c_oflag* of the **termios** structure.

opost (**-opost**)
Post-process output (do not post-process output; ignore all other output modes).

onlcr (**-onlcr**)
Map (do not map) NL to CR-NL on output.

oxtabs (**-oxtabs**)
Expand (do not expand) tabs to spaces on output.

Local Modes

Local mode flags (lflags) or ‘‘line discipline flags’’ affect various and sundry characteristics of terminal processing, and which correspond to the *c_lflag* of the *termios* structure.

isig (**-isig**)
Enable (disable) the checking of characters against the special control characters INTR , QUIT , and SUSP .

icanon (**-icanon**)
Enable (disable) canonical input (ERASE and KILL processing).

iexten (**-iexten**)
Enable (disable) any implementation defined special control characters not currently controlled by icanon, isig, or ixon.

echo (**-echo**)
Echo back (do not echo back) every character typed.

echoe (**-echoe**)
The ERASE character shall (shall not) visually erase the last character

in the current line from the display, if possible.

echok (**-echok**)
Echo (do not echo) NL after KILL character.

echoke (**-echoke**)
The KILL character shall (shall not) visually erase the the current line from the display, if possible.

echonl (**-echonl**)
Echo (do not echo) NL , even if echo is disabled.

echoctl (**-echoctl**)
If *echoctl* is set, echo control characters as ^X. Otherwise control characters echo as themselves.

echoprt (**-echoprt**)
For printing terminals. If set, echo erased characters backwards within ''\'' and ''/''. Otherwise, disable this feature.

noflsh (**-noflsh**)
Disable (enable) flush after INTR , QUIT , SUSP .

tostop (**-tostop**)
Send (do not send) SIGTTOU for background output. This causes background jobs to stop if they attempt terminal output. The default behavior is **–tostop**.

altwerase (**-altwerase**)
Use (do not use) an alternate word erase algorithm when processing WERASE characters. This alternate algorithm considers sequences of alphanumeric / underscores as words. It also skips the first preceding character in its classification (as a convenience since the one preceding character could have been erased with simply an ERASE character.)

mdmbuf (**-mdmbuf**)
If set, flow control output based on condition of Carrier Detect. Otherwise writes return an error if Carrier Detect is low (and Carrier is not being ignored with the CLOCAL flag.)

flusho (**-flusho**)
Indicates output is (is not) being discarded.

pendin (**-pendin**)
Indicates input is (is not) pending after a switch from non-canonical to canonical mode and will be re-input when a read becomes pending or more input arrives.

Control Characters

control-character string
Set *control-character* to *string*. If string is a single character, the control character is set to that character. If *string* is the two character sequence "^-" or the string "undef", the control character is disabled (i.e. set to {_POSIX_VDISABLE}. Setting the control character also sets the corresponding element of the **termios c_cc** array.

belltime *number*
After a bell sounds at the keyboard, any subsequent bells delivered in the next *number* 10ths of a second are ignored. This has no effect on other characters.

min *number*
time *number*

Set the value of **min** or **time** to *number*. MIN and TIME are used in non-canonical mode input processing (**-icanon**).

The recognized control-characters are:

Control-character	Subscript	Description	Default
belltime	VBELTIME	VBELTIME value	3
dsusp	VDSUSP	DSUSP	No character
eof	VEOF	EOF	^D
eol2	VEOL2	EOL2	No character
eol	VEOL	EOL	No character
erase	VERASE	ERASE	No character
intr	VINTR	INTR	No character
kill	VKILL	KILL	^U
lnext	VLNEXT	LNEXT	No character
min	VMIN	MIN value	
quit	VQUIT	QUIT	No character
reprint	VREPRINT	REPRINT	No character
start	VSTART	START	No character
status	VSTATUS	STATUS	No character
stop	VSTOP	STOP	No character
susp	VSUSP	SUSP	^Z
time	VTIME	TIME value	
werase	VWERASE	WERASE	No character

Combination Modes

saved settings
Set the current terminal characteristics to the saved settings produced by the **–g** option.

evenp or **parity**
Enable parenb and cs7; disable parodd.

oddp Enable **parenb**, **cs7**, and **parodd**.

-parity,
Disable parenb, and set cs8.

nl (**-nl**) Enable (disable) icrnl. In addition **–nl** unsets inlcr and igncr.

ek Reset ERASE and KILL characters back to system defaults.

sane Resets all modes to reasonable values for interactive terminal use.

tty Set the line discipline to the standard terminal line discipline TTYDISC .

crt (**-crt**)
Set (disable) all modes suitable for a CRT display device.

kerninfo (**-kerninfo**)
Enable (disable) the system generated status line associated with processing a STATUS character (usually set to ^T). The status line consists of the system load average, the current command name, its process ID, the event the process is waiting on (or the status of the process), the user and system times, percent cpu, and current memory usage.

columns *number*
The terminal size is recorded as having *number* columns. On an

INTERIX console window, this will resize the terminal window to have *number* columns, unless the new size is larger than the physical screen.

cols *number*
is an alias for **columns**.

rows *number*
The terminal size is recorded as having *number* rows. On an INTERIX console window, this will resize the terminal window to have *number* rows, unless the new size is larger than the physical screen.

dec Set modes suitable for users of Digital Equipment Corporation systems (ERASE , KILL , and INTR characters are set to ^?, ^U, and ^C; ixany is disabled, and crt is enabled.)

extproc (**-extproc**)
If set, this flag indicates that some amount of terminal processing is being performed by either the terminal hardware or by the remote side connected to a pty.

raw (**-raw**)
If set, change the modes of the terminal so that no input or output processing is performed. If unset, change the modes of the terminal to some reasonable state that performs input and output processing. Note that since the terminal driver no longer has a single RAW bit, it is not possible to intuit what flags were set prior to setting *raw*. This means that unsetting *raw* may not put back all the setting that were previously in effect. To set the terminal into a raw state and then accurately restore it, the following shell code is recommended:

```
save_state=$(stty -g)
stty raw
...
stty "$save_state"
```

size The size of the terminal is printed as two numbers on a single line, first rows, then columns.

Compatibility Modes

These modes remain for compatibility with the previous version of the *stty* command.

all Reports all the terminal modes as with

```
stty -a
```

except that the control characters are printed in a columnar format.

everything
Same as **all**.

cooked Same as **sane**.

cbreak If set, enables **brkint**, **ixon**, **imaxbel**, **opost**, **isig**, **iexten**, and **icanon**. If unset, same as **sane**.

new Same as **tty**.

old Same as **tty**.

newcrt (**-newcrt**)
Same as **crt**.

pass8 The converse of **parity**.

tandem (**-tandem**)

Same as **ixoff**.

decctlq (**-decctlq**)
Same as **ixany**.

crterase (**-crterase**)
Same as **echoe**.

crtbs (**-crtbs**)
Same as **echoe**.

crtkill (**-crtkill**)
Same as **echoke**.

ctlecho (**-ctlecho**)
Same as **echoctl**.

prterase (**-prterase**)
Same as **echoprt**.

litout (**-litout**)
The converse of **opost**.

tabs (**-tabs**)
The converse of **tabs**.

brk (**-value**)
Same as the control character **eol**.

flush (**-value**)
Same as the control character **discard**.

rprnt (**-value**)
Same as the control character **reprint**.

DIAGNOSTICS

The *stty* utility exits with a value of 0 if successful, and >0 if an error occurs.

SEE ALSO

tty

NAME

syslogd – log systems messages

SYNOPSIS

`syslogd [-ds] [-f config_file] [-m mark_interval]`

DESCRIPTION

Syslogd reads and logs messages to the system console, log files, other machines and / or users as specified by its configuration file. The options are as follows:

–d Run in debug mode.

–f Specify the pathname of an alternate configuration file; the default is **$OPENNT_ROOT/etc/syslog.conf**.

–m Select the number of minutes between ''mark'' messages; the default is 20 minutes.

–s Run as a service (do not fork upon startup). This option is *required* to run *syslogd* as a Windows NT service.

Syslogd reads its configuration file when it starts up and whenever it receives a hangup signal. For information on the format of the configuration file, see *syslog.conf*.

Syslogd from an Internet domain socket. *Syslogd* creates the file **$OPENNT_ROOT/var/run/syslog.pid**, and stores its process ID there. This can be used to kill or reconfigure *syslogd*.

The message sent to *syslogd* should consist of a single line. The message can contain a priority code, which should be a preceding decimal number in angle braces, for example, **<5>**. This priority code should map into the priorities defined in the include file **<sys/syslog.h>**.

FILES

$OPENNT_ROOT/etc/syslog.conf
The configuration file.

$OPENNT_ROOT/var/run/syslog.pid
The process ID of current *syslogd*.

SEE

logger

syslog()

syslog.conf

NAME

tail – display the last part of a file

SYNOPSIS

`tail [-f] [-b` *number* `| -c` *number* `| -n` *number*`] [`*file ...*`]`

DESCRIPTION

The *tail* utility displays the contents of *file* or, by default, its standard input, to the standard output.

The display begins at a byte, line or 512-byte block location in the input. Numbers having a leading plus (‘‘+’’) sign are relative to the beginning of the input, for example, ‘‘-c +2’’ starts the display at the second byte of the input. Numbers having a leading minus (‘‘-’’) sign or no explicit sign are relative to the end of the input, for example, ‘‘-n 2’’ displays the last two lines of the input. The default starting location is ‘‘-n 10’’, or the last 10 lines of the input.

The options are as follows:

–b *number*
The location is *number* 512-byte blocks.

–c *number*
The location is *number* bytes.

–f The **–f** option causes *tail* to not stop when end of file is reached, but rather to wait for additional data to be appended to the input. The **–f** option is ignored if the standard input is a pipe, but not if it is a FIFO.

–n *number*
The location is *number* lines.

If more than a single file is specified, each file is preceded by a header consisting of the string ‘‘==> XXX <==’’ where ‘‘XXX’’ is the name of the file.

DIAGNOSTICS

The *tail* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

cat, *head*, *sed*

NAME

talk, talkd – talk to another user

SYNOPSIS

`talk` *`address`* **[***`terminal`***]**

`talkd`

DESCRIPTION

Talk is a two-way, screen-oriented communications program; you can communicate with the user specified by *address*, which is the other user's login name. On INTERIX, do not specify a domain as part of the address.

If the recipient is logged in to more than one terminal, the message appears on all of the accessible terminals, unless you specify a particular terminal as the *terminal* argument. (The format of *terminal* is the same as the TTY field from the output of *who*).

The talk protocol is case-independent and checks only the first 11 characters of the name. If the recipient is ambiguous (the login names are identical in the first 11 characters, although the domains may be different or the same), specify the *terminal* to identify the correct recipient.

When you start *talk*, it sends a message like this one to the recipient:

```
Message from your name
talk: connection requested by your_address
talk: respond with: talk your_address
```

To establish communication, the recipient types the command

```
$ talk your_address
```

Once communication is established, both users can type their messages simultaneously with their output displayed in different parts of the screen. Special commands are:

Alert recipient's terminal	Alert character (usually Control-G)
Refresh sender's screen regions	Control-L
End session	EOF (usually Control-D); both sides must give the command
Backspace	Erase character (usually Control-H or backspace)
Clear line	Kill character (usually Control-U)

In order for *talk* to run, the system must be configured to run *ntalkd* through the *inetd* service; see the *inetd* reference page.

DIAGNOSTICS

On successful completion, *talk* returns an exit status of 0. If an error occurred, it returns >0.

SEE ALSO

ntalkd, *stty*, *who*

NAME

tar – create archive files

SYNOPSIS

```
tar c [befHLlmoPpvwX] [B bytes] [file ...]
tar r [befHLlmoPpvwX] [B bytes] [file ...]
tar x [befHLlmopvwX] [file ...]
tar t [befHLlmopvwX] [file ...]
tar u [befHLlmopvwX] [B bytes] [file ...]
```

DESCRIPTION

The INTERIX products provide the *pax* utility to read and write *tar*-format files. *Pax* can read and write files in both the POSIX.2 standard format (USTAR) and the old BSD4.3 format. If *pax* is invoked as *tar*, it behaves in the same manner as *tar*. For your convenience, the *tar* options are presented here. For more information on the archive format and the behavior of the utility, see the *pax* reference page.

The *tar* command takes as arguments a *function* letter followed by zero or more modifying letters. Each function is given its own synopsis above; the functions are:

c **Create** the new archive; writes specified files starting at the beginning of the archive instead of appending after the end.

r **Write** the specified *file* or files to the end of the archive.

t **Table of contents** Write the archive's table of contents to standard output.

u **Update** the archive by adding *file* or files if they are not already there, or if they have been modified since they were added to the archive.

x **Extract** the named *file* or files from the archive. If *file* is a directory, its contents are extracted recursively.

These functions can be modified by the following letters. Not all functions accept all modifiers.

b Use the first *file* (or the second, if **f** has already been specified) as the blocking factor for tape records. Only use this modifier with raw magnetic tape archives.

B *bytes* Limit the number of bytes written to a single archive volume to *bytes*. The *bytes* limit can end with **m**, **k**, or **b** to specify multiplication by 1048576 (1M), 1024 (1K) or 512, respectively. A pair of *bytes* limits can be separated by *x* to indicate a product.

Warning: Only use this option when writing an archive to a device which supports an end-of-file read condition based on last (or largest) write offset (such as a regular file or a tape drive). The use of this option with a floppy or hard disk is not recommended.

e Stop at the first error.

f Use the first *file* operand (or the second if **b** has already been specified) as the name of the archive. If *file* is –, read from standard input or write to standard output, whichever is appropriate.

H Follow command-line symbolic links only. Since INTERIX does not support symbolic links, this option has no effect.

l Report if links to the files being archived cannot be resolved.

L Follow symbolic links. Since INTERIX does not support symbolic links, this option has no effect.

m Do not restore the file modification times; instead, each file is marked with the time of extraction.

o Give the extracted files the user and group identifiers of the user running *tar*, rather than trying to preserve the original values.

p Preserve user ID, group ID, file mode, access and modification times.

P Do not follow symbolic links. Since INTERIX does not support symbolic links, this is the default.

v Be verbose; write the operation and the name of each file processed to standard error. For the functions **c**, **r**, and **u**, *tar* writes "a" as the operation; for the function **x**, *tar* writes "x".

w Wait for confirmation before each action.

X Do not pass over mount points in the file system. This option has no effect on INTERIX.

ENVIRONMENT VARIABLES

Tar makes use of the following environment variable, if defined:

TAPE Contains the default archive name, used if the **–f** option is not specified.

NOTES

Sometimes when creating or writing to an archive, the following message is printed to standard error:

```
uid nnnnnnn too large to fit in header: substituting 32666
```

On traditional systems, the user ID and group ID numbers fit into 16 bits, but on INTERIX, they can be 32 bits long. The INTERIX uids and gids don't fit into the **tar** or **ustar** archive formats. This implementation substitutes the value 32666 (decimal) for the original uid or gid.

As a consequence, you can get errors when unpacking tar archives you created because they do not contain your uid. The solution is to use **ustar** format. Substitution isn't a problem with the **ustar** format. That format records both the numeric and character string representations of these numbers, and the string representation takes precedence over the numeric.

SEE ALSO

pax

NAME

tcsh – C shell with file name completion and command line editing

SYNOPSIS

```
tcsh [-bcdefFimnqstvVxX] [-Dname[=value]] [arg ...]
tcsh -l
```

DESCRIPTION

tcsh is an enhanced but completely compatible version of the BSD C shell, *csh*. It is a command language interpreter usable both as an interactive login shell and a shell script command processor. It includes a command-line editor (see "The command-line editor"), programmable word completion (see "Completion and listing"), spelling correction (see "Spelling correction"), a history mechanism (see "History substitution"), job control (see "Jobs") and a C-like syntax. The "NEW FEATURES" section describes major enhancements of *tcsh* over *csh*. Throughout this manual, features of *tcsh* not found in most *csh* implementations (specifically, the 4.4BSD *csh*) are labeled with '(+)', and features which are present in *csh* but not usually documented are labeled with '(u)'.

Argument list processing

If the first argument (argument 0) to the shell is '–' then it is a login shell. A login shell can be also specified by invoking the shell with the **–l** flag as the only argument.

The rest of the flag arguments are interpreted as follows:

–b Force a ''break'' from option processing, causing any further shell arguments to be treated as non-option arguments. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion or possible subterfuge. The shell will not run a set-user ID script without this option.

–c Read commands from the following argument (which must be present, and must be a single argument), stored in the *command* shell variable for reference, and executed. Any remaining arguments are placed in the *argv* shell variable.

–d Load the directory stack from **~/.cshdirs** as described under "Startup and shutdown", whether or not it is a login shell. (+)

–e Exit if any invoked command terminates abnormally or yields a non-zero exit status.

–f Fast start; ignore **~/.tcshrc** to start faster.

–i Interactive shell; prompt for top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

–l The shell is a login shell. Only applicable if **–l** is the only flag specified.

–m Load **~/.tcshrc** even if it does not belong to the effective user. Newer versions of *su* can pass **–m** to the shell. (+)

–n Parse commands but do not execute them. This aids in debugging shell scripts.

–q Accept SIGQUIT (see "Signal handling"), and behave when it is used under a debugger. Job control is disabled. (u)

–s Take command input from the standard input.

–t Read and execute a single line of input. A '\' may be used to escape the newline at the end of this line and continue onto another line.

–v Set the *verbose* shell variable so that command input is echoed after history substitution.

–V Set the *verbose* shell variable even before executing **~/.tcshrc**.

–x Set the *echo* shell variable, so that commands are echoed immediately before execution.

–X Set the *echo* shell variable even before executing **~/.tcshrc**.

After processing of flag arguments, if arguments remain but none of the **–c**, **–i**, **–s**, or **–t** options were given, the first argument is taken as the name of a file of commands, or "script", to be executed. The shell opens this file and saves its name for possible resubstitution by '$0'. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell uses such a 'standard' shell to execute a script whose first character is not a '#', i.e. which does not start with a comment.

Remaining arguments are placed in the *argv* shell variable.

Startup and shutdown

A login shell begins by executing commands from the system files **$OPENNT_ROOT/etc/csh.cshrc** and **$OPENNT_ROOT/etc/csh.login**. It then executes commands from files in the user's *home* directory: first **~/.tcshrc** (+) or, if **~/.tcshrc** is not found, **~/.cshrc**, then **~/.history** (or the value of the *histfile* shell variable), then **~/.login**, and finally **~/.cshdirs** (or the value of the *dirsfile* shell variable) (+). The shell may read **$OPENNT_ROOT/etc/csh.login** before instead of after **$OPENNT_ROOT/etc/csh.cshrc**, and **~/.login** before instead of after **~/.tcshrc** or **~/.cshrc** and **~/.history**, if so compiled; see the *version* shell variable. (+)

Non-login shells read only **$OPENNT_ROOT/etc/csh.cshrc** and **~/.tcshrc** or **~/.cshrc** on startup.

Commands like *stty* and *tset* which need be run only once per login, usually go in one's **~/.login** file. Users who need to use the same set of files with both *csh* and *tcsh* can have only a **~/.cshrc** which checks for the existence of the *tcsh* shell variable (q.v.) before using *tcsh*-specific commands, or can have both a **~/.cshrc** and a **~/.tcshrc** which **source**s (see the builtin command) **~/.cshrc**. The rest of this manual uses "**~/.tcshrc**" to mean "**~/.tcshrc** or, if **~/.tcshrc** is not found, **~/.cshrc**."

In the normal case, the shell begins reading commands from the terminal, prompting with '> '. (Processing of arguments and the use of the shell to process files containing command scripts are described later.) The shell repeatedly reads a line of command input, breaks it into words, places it on the command history list, parses it and executes each command in the line.

One can log out by typing '^D' on an empty line, 'logout' or 'login' or via the shell's autologout mechanism (see the *autologout* shell variable). When a login shell terminates it sets the *logout* shell variable to 'normal' or 'automatic' as appropriate, then executes commands from the files **$OPENNT_ROOT/etc/csh.logout** and **~/.logout**. The shell may drop DTR on logout if so compiled; see the *version* shell variable.

The names of the system login and logout files vary from system to system for compatibility with different *csh* variants; see "FILES."

Editing

We first describe **The command-line editor**. The "Completion and listing" and "Spelling correction" sections describe two sets of functionality which are implemented as editor commands but which deserve their own treatment. Finally, "Editor commands" lists and describes the editor commands specific to the shell and their default bindings.

The command-line editor (+)

Command-line input can be edited using key sequences much like those used in GNU Emacs or *vi*. The editor is active only when the *edit* shell variable is set, which it is by default in interactive shells. The *bindkey* builtin can display and change key bindings. Vi-style key bindings are used by default in the INTERIX version of the shell. On other systems, Emacs-style bindings may be the default, depending upon how the shell was compiled; (unless the shell was compiled otherwise; see the *version* shell variable. **Bindkey** can change the key bindings to *emacs*-style bindings en masse.

The shell always binds the arrow keys (as defined in the *TERMCAP* environment variable) to

down	*down-history*
up	*up-history*
left	*backward-char*
right	*forward-char*

unless doing so would alter another single-character binding. One can set the arrow key escape sequences to the empty string with **settc** to prevent these bindings. The ANSI/VT100 sequences for arrow keys are always bound.

Other key bindings are, for the most part, what Emacs and *vi* users would expect and can easily be displayed by **bindkey**, so there is no need to list them here. Likewise, **bindkey** can list the editor commands with a short description of each.

Note that editor commands do not have the same notion of a "word" as does the shell. The editor delimits words with any non-alphanumeric characters not in the shell variable *wordchars*, while the shell recognizes only whitespace and some of the characters with special meanings to it, listed under "Lexical

structure."

Completion and listing (+)

The shell is often able to complete words when given a unique abbreviation. Type part of a word (for example 'ls /usr/lost') and hit the tab key to run the **complete-word** editor command. The shell completes the filename '/usr/lost' to '/usr/lost+found/', replacing the incomplete word with the complete word in the input buffer. (Note the terminal '/'; completion adds a '/' to the end of completed directories and a space to the end of other completed words, to speed typing and provide a visual indicator of successful completion. The *addsuffix* shell variable can be unset to prevent this.) If no match is found (perhaps '/usr/lost+found' doesn't exist), the terminal bell rings. If the word is already complete (perhaps there is a '/usr/lost' on your system, or perhaps you were thinking too far ahead and typed the whole thing) a '/' or space is added to the end if it isn't already there.

Completion works anywhere in the line, not just at the end; completed text pushes the rest of the line to the right. Completion in the middle of a word often results in leftover characters to the right of the cursor which need to be deleted.

Commands and variables can be completed in much the same way. For example, typing 'em[tab]' would complete 'em' to 'emacs' if *emacs* were the only command on your system beginning with 'em'. Completion can find a command in any directory in *path* or if given a full pathname. Typing 'echo $ar[tab]' would complete '$ar' to '$argv' if no other variable began with 'ar'.

The shell parses the input buffer to determine whether the word you want to complete should be completed as a filename, command or variable. The first word in the buffer and the first word following ';', '|', '|&', '&&' or '||' is considered to be a command. A word beginning with '$' is considered to be a variable. Anything else is a filename. An empty line is 'completed' as a filename.

You can list the possible completions of a word at any time by typing '^D' to run the **delete-char-or-list-or-eof** editor command. The shell lists the possible completions using the *ls–F* builtin (q.v.) and reprints the prompt and unfinished command line, for example:

```
> ls /usr/l[^D]
lbin/       lib/        local/      lost+found/
> ls /usr/l
```

If the *autolist* shell variable is set, the shell lists the remaining choices (if any) whenever completion fails:

```
> set autolist
> nm /usr/lib/libt[tab]
libtermcap.a@ libtermlib.a@
> nm /usr/lib/libterm
```

If *autolist* is set to 'ambiguous', choices are listed only when completion fails and adds no new characters to the word being completed.

A filename to be completed can contain variables, your own or others' home directories abbreviated with '~' (see "Filename substitution") and directory stack entries abbreviated with '=' (see "Directory stack substitution"). For example,

```
> ls ~k[^D]
kahn    kas     kellogg
> ls ~ke[tab]
> ls ~kellogg/
```

or

```
> set local = /usr/local
> ls $lo[tab]
> ls $local/[^D]
bin/ etc/ lib/ man/ src/
> ls $local/
```

Note that variables can also be expanded explicitly with the **expand-variables** editor command.

delete-char-or-list-or-eof only lists at the end of the line; in the middle of a line it deletes the character under the cursor and on an empty line it logs one out or, if *ignoreeof* is set, does nothing. 'M-^D', bound to the editor command **list-choices**, lists completion possibilities anywhere on a line, and **list-choices** (or any one of the related editor commands which do or don't delete, list and/or log out, listed under **delete-char-or-list-or-eof**) can be bound to '^D' with the **bindkey** builtin command if so desired.

The **complete-word-fwd** and **complete-word-back** editor commands (not bound to any keys by default) can be used to cycle up and down through the list of possible completions, replacing the current word with the next or previous word in the list.

The shell variable *fignore* can be set to a list of suffixes to be ignored by completion. Consider the following:

```
> ls
Makefile        condiments.h~   main.o    side.c
README          main.c         meal      side.o
condiments.h    main.c~
> set fignore = (.o \~)
> emacs ma[^D]
main.c   main.c~  main.o
> emacs ma[tab]
> emacs main.c
```

'main.c~' and 'main.o' are ignored by completion (but not listing), because they end in suffixes in *fignore*. Note that a '\' was needed in front of '~' to prevent it from being expanded to *home* as described under "Filename substitution". *fignore* is ignored if only one completion is possible.

If the *complete* shell variable is set to 'enhance', completion 1) ignores case and 2) considers periods, hyphens and underscores ('.', '–' and '_') to be word separators and hyphens and underscores to be equivalent. If you had the following files

```
comp.lang.c      comp.lang.perl   comp.std.c++
comp.lang.c++    comp.std.c
```

and typed 'mail –f c.l.c[tab]', it would be completed to 'mail –f comp.lang.c', and ^D would list 'comp.lang.c' and 'comp.lang.c++'. 'mail –f c..c++[^D]' would list 'comp.lang.c++' and 'comp.std.c++'. Typing 'rm a—file[^D]' in the following directory

```
A_silly_file  a-hyphenated-file  another_silly_file
```

would list all three files, because case is ignored and hyphens and underscores are equivalent. Periods, however, are not equivalent to hyphens or underscores.

Completion and listing are affected by several other shell variables: *recexact* can be set to complete on the shortest possible unique match, even if more typing might result in a longer match:

```
> ls
fodder   foo      food     foonly
> set recexact
> rm fo[tab]
```

just beeps, because 'fo' could expand to 'fod' or 'foo', but if we type another 'o',

```
> rm foo[tab]
> rm foo
```

the completion completes on 'foo', even though 'food' and 'foonly' also match. *autoexpand* can be set to run the **expand-history** editor command before each completion attempt, *autocorrect* can be set to spelling-correct the word to be completed (see "Spelling correction") before each completion attempt and *correct* can be set to complete commands automatically after one hits 'return'. *matchbeep* can be set to make completion beep or not beep in a variety of situations, and *nobeep* can be set to never beep at all. *nostat* can be set to a list of directories and/or patterns which match directories to prevent the completion mechanism from *stat*()ing those directories. *listmax* and *listmaxrows* can be set to limit the number of items and rows (respectively) that are listed without asking first. **recognize_only_executables** can be set to make the shell list only executables when listing commands, but it is quite slow.

Finally, the **complete** builtin command can be used to tell the shell how to complete words other than filenames, commands and variables. Completion and listing do not work on glob-patterns (see "Filename substitution"), but the **list-glob** and **expand-glob** editor commands perform equivalent functions for glob-patterns.

Spelling correction (+)

The shell can sometimes correct the spelling of filenames, commands and variable names as well as completing and listing them.

Individual words can be spelling-corrected with the **spell-word** editor command (usually bound to M-s and M-S) and the entire input buffer with **spell-line** (usually bound to M-$). The *correct* shell variable can be set to 'cmd' to correct the command name or 'all' to correct the entire line each time return is typed, and *autocorrect* can be set to correct the word to be completed before each completion attempt.

When spelling correction is invoked in any of these ways and the shell thinks that any part of the command line is misspelled, it prompts with the corrected line:

```
> set correct = cmd
> lz /usr/bin
CORRECT>ls /usr/bin (y|n|e|a)?
```

One can answer 'y' or space to execute the corrected line, 'e' to leave the uncorrected command in the input buffer, 'a' to abort the command as if '^C' had been hit, and anything else to execute the original line unchanged.

Spelling correction recognizes user-defined completions (see the **complete** builtin command). If an input word in a position for which a completion is defined resembles a word in the completion list, spelling correction registers a misspelling and suggests the latter word as a correction. However, if the input word does not match any of the possible completions for that position, spelling correction does not register a misspelling.

Like completion, spelling correction works anywhere in the line, pushing the rest of the line to the right and possibly leaving extra characters to the right of the cursor.

Beware: spelling correction is not guaranteed to work the way one intends, and is provided mostly as an experimental feature. Suggestions and improvements are welcome.

Editor commands (+)

'bindkey' lists key bindings and 'bindkey –l' lists and briefly describes editor commands. Only new or especially interesting editor commands are described here. See *emacs* and *vi* for descriptions of each editor's key bindings.

The character or characters to which each command is bound by default is given in parentheses. '^*character*' means a control character and 'M-*character*' a meta character, typed as escape-*character* on terminals without a meta key. Case counts, but commands which are bound to letters by default are bound to both lower- and uppercase letters for convenience.

complete-word (tab)
: Completes a word as described under "Completion and listing".

complete-word-back (not bound)
: Like **complete-word-fwd**, but steps up from the end of the list.

complete-word-fwd (not bound)
: Replaces the current word with the first word in the list of possible completions. May be repeated to step down through the list. At the end of the list, beeps and reverts to the incomplete word.

complete-word-raw (^X-tab)
: Like **complete-word**, but ignores user-defined completions.

copy-prev-word (M-^_)
: Copies the previous word in the current line into the input buffer. See also **insert-last-word**.

dabbrev-expand (M-/)
: Expands the current word to the most recent preceding one for which the current is a leading substring, wrapping around the history list

(once) if necessary. Repeating **dabbrev-expand** without any intervening typing changes to the next previous word etc., skipping identical matches much like **history-search-backward** does.

delete-char (not bound)
Deletes the character under the cursor. See also **delete-char-or-list-or-eof**.

delete-char-or-eof (not bound)
Does **delete-char** if there is a character under the cursor or **end-of-file** on an empty line. See also **delete-char-or-list-or-eof**.

delete-char-or-list (not bound)
Does **delete-char** if there is a character under the cursor or **list-choices** at the end of the line. See also **delete-char-or-list-or-eof**.

delete-char-or-list-or-eof (^D)
Does **delete-char** if there is a character under the cursor, **list-choices** at the end of the line or **end-of-file** on an empty line. See also those three commands, each of which only does a single action, and **delete-char-or-eof**, **delete-char-or-list** and **list-or-eof**, each of which does a different two out of the three.

down-history (down-arrow, ^N)
Like **up-history**, but steps down, stopping at the original input line.

end-of-file (not bound)
Signals an end of file, causing the shell to exit unless the *ignoreeof* shell variable (q.v.) is set to prevent this. See also **delete-char-or-list-or-eof**.

expand-history (M-space)
Expands history substitutions in the current word. See "History substitution". See also **magic-space**, **toggle-literal-history** and the *autoexpand* shell variable.

expand-glob (^X-*)
Expands the glob-pattern to the left of the cursor. See "Filename substitution".

expand-line (not bound)
Like **expand-history**, but expands history substitutions in each word in the input buffer,

expand-variables (^X-$)
Expands the variable to the left of the cursor. See "Variable substitution".

history-search-backward (M-p, M-P)
Searches backwards through the history list for a command beginning with the current contents of the input buffer up to the cursor and copies it into the input buffer. The search string may be a glob-pattern (see "Filename substitution") containing '*', '?', '[]' or '{}'. **up-history** and **down-history** will proceed from the appropriate point in the history list. Emacs mode only. See also **history-search-forward** and **i-search-back**.

history-search-forward (M-n, M-N)
Like **history-search-backward**, but searches forward.

i-search-back (not bound)
Searches backward like **history-search-backward**, copies the first match into the input buffer with the cursor positioned at the end of the pattern, and prompts with 'bck: ' and the first match. Additional characters may be typed to extend the search, **i-search-back** may be

typed to continue searching with the same pattern, wrapping around the history list if necessary (**i-search-back** must be bound to a single character for this to work), or one of the following special characters may be typed:

^W Appends the rest of the word under the cursor to the search pattern.

delete (or any character bound to **backward-delete-char**)
Undoes the effect of the last character typed and deletes a character from the search pattern if appropriate.

^G If the previous search was successful, aborts the entire search. If not, goes back to the last successful search.

escape Ends the search, leaving the current line in the input buffer.

Any other character not bound to **self-insert-command** terminates the search, leaving the current line in the input buffer, and is then interpreted as normal input. In particular, a carriage return causes the current line to be executed. Emacs mode only. See also **i-search-fwd** and **history-search-backward**.

i-search-fwd (not bound)
Like **i-search-back**, but searches forward.

insert-last-word (M-_)
Inserts the last word of the previous input line ('!$') into the input buffer. See also **copy-prev-word**.

list-choices (M-^D)
Lists completion possibilities as described under "Completion and listing". See also **delete-char-or-list-or-eof** and **list-choices-raw**.

list-choices-raw (^X-^D)
Like **list-choices**, but ignores user-defined completions.

list-glob (^X-g, ^X-G)
Lists (via the **ls–F** builtin) matches to the glob-pattern (see "Filename substitution") to the left of the cursor.

list-or-eof (not bound)
Does **list-choices** or **end-of-file** on an empty line. See also **delete-char-or-list-or-eof**.

magic-space (not bound)
Expands history substitutions in the current line, like **expand-history**, and appends a space. **magic-space** is designed to be bound to the spacebar, but is not bound by default.

normalize-command (^X-?)
Searches for the current word in PATH and, if it is found, replaces it with the full path to the executable. Special characters are quoted. Aliases are expanded and quoted but commands within aliases are not. This command is useful with commands which take commands as arguments, e.g. 'dbx' and 'sh –x'.

normalize-path (^X-n, ^X-N)
Expands the current word as described under the 'expand' setting of the *symlinks* shell variable.

overwrite-mode (unbound)
Toggles between input and overwrite modes.

run-fg-editor (M-^Z)
Saves the current input line and looks for a stopped job with a name

equal to the last component of the file name part of the *EDITOR* or *VISUAL* environment variables, or, if neither is set, 'ed' or 'vi'. If such a job is found, it is restarted as if 'fg %**job**' had been typed. This is used to toggle back and forth between an editor and the shell easily. Some people bind this command to '^Z' so they can do this even more easily.

run-help (M-h, M-H)
Searches for documentation on the current command, using the same notion of 'current command' as the completion routines, and prints it. There is no way to use a pager; **run-help** is designed for short help files. Documentation should be in a file named *command*.help, *command*.1, *command*.6, *command*.8 or *command*, which should be in one of the directories listed in the *HPATH* environment variable. If there is more than one help file only the first is printed.

self-insert-command (text characters)
In insert mode (the default), inserts the typed character into the input line after the character under the cursor. In overwrite mode, replaces the character under the cursor with the typed character. The input mode is normally preserved between lines, but the *inputmode* shell variable can be set to 'insert' or 'overwrite' to put the editor in that mode at the beginning of each line. See also **overwrite-mode**.

sequence-lead-in (arrow prefix, meta prefix, ^X)
Indicates that the following characters are part of a multi-key sequence. Binding a command to a multi-key sequence really creates two bindings: the first character to **sequence-lead-in** and the whole sequence to the command. All sequences beginning with a character bound to **sequence-lead-in** are effectively bound to **undefined-key** unless bound to another command.

spell-line (M-$)
Attempts to correct the spelling of each word in the input buffer, like **spell-word**, but ignores words whose first character is one of '!', '.', '\', '^', '–', '*' or '%' to avoid problems with switches, substitutions and the like. See "Spelling correction".

spell-word (M-s, M-S)
Attempts to correct the spelling of the current word as described under "Spelling correction". Checks each component of a word which appears to be a pathname.

toggle-literal-history (M-r, M-R)
Expands or 'unexpands' history substitutions in the input buffer. See also **expand-history** and the *autoexpand* shell variable.

undefined-key (any unbound key)
Beeps.

up-history (up-arrow, ^P)
Copies the previous entry in the history list into the input buffer. If *histlit* is set, uses the literal form of the entry. May be repeated to step up through the history list, stopping at the top.

vi-search-back (?)
Prompts with '?' for a search string (which may be a glob-pattern, as with **history-search-backward),** searches for it and copies it into the input buffer. The bell rings if no match is found. Hitting return ends the search and leaves the last match in the input buffer. Hitting

escape ends the search and executes the match. **vi** mode only.

vi-search-fwd (/)
Like **vi-search-back**, but searches forward.

which-command (M-?)
Does a **which** (see the description of the builtin command) on the first word of the input buffer.

Lexical structure

The shell splits input lines into words at blanks and tabs. The special characters '&', '|', ';', '<', '>', '(', and ')' and the doubled characters '&&', '||', '<<' and '>>' are always separate words, whether or not they are surrounded by whitespace.

When the shell's input is not a terminal, the character '#' is taken to begin a comment. Each '#' and the rest of the input line on which it appears is discarded before further parsing.

A special character (including a blank or tab) may be prevented from having its special meaning, and possibly made part of another word, by preceding it with a backslash ('\') or enclosing it in single (''), double ('"') or backward ('`') quotes. When not otherwise quoted a newline preceded by a '\' is equivalent to a blank, but inside quotes this sequence results in a newline.

Furthermore, all "Substitutions" (see below) except "History substitution" can be prevented by enclosing the strings (or parts of strings) in which they appear with single quotes or by quoting the crucial character(s) (e.g. '$' or '`' for "Variable substitution" or "Command substitution" respectively) with '\'. "History substitution"s are quoted in the same way by backslashes but not by single quotes. Strings quoted with double or backward quotes undergo "Variable substitution" and "Command substitution", but other substitutions are prevented.

Text inside single or double quotes becomes a single word (or part of one). Metacharacters in these strings, including blanks and tabs, do not form separate words. Only in one special case (see "Command substitution" below) can a double-quoted string yield parts of more than one word; single-quoted strings never do. Backward quotes are special: they signal "Command substitution" (q.v.), which may result in more than one word.

Quoting complex strings, particularly strings which themselves contain quoting characters, can be confusing. Remember that quotes need not be used as they are in human writing! It may be easier to quote not an entire string, but only those parts of the string which need quoting, using different types of quoting to do so if appropriate.

The *backslash_quote* shell variable (q.v.) can be set to make backslashes always quote '\', ''', and '"'. (+) This may make complex quoting tasks easier, but it can cause syntax errors in *csh* scripts.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur. We note in passing the data structures involved and the commands and variables which affect them. Remember that substitutions can be prevented by quoting as described under "Lexical structure".

History substitution

Each command, or ''event'', input from the terminal is saved in the history list. The previous command is always saved, and the *history* shell variable can be set to a number to save that many commands. The *histdup* shell variable can be set to not save duplicate events or consecutive duplicate events.

Saved commands are numbered sequentially from 1 and stamped with the time. It is not usually necessary to use event numbers, but the current event number can be made part of the prompt by placing an '!' in the *prompt* shell variable.

The shell actually saves history in expanded and literal (unexpanded) forms. If the *histlit* shell variable is set, commands that display and store history use the literal form.

The **history** builtin command can print, store in a file, restore and clear the history list at any time, and the *savehist* and *histfile* shell variables can be can be set to store the history list automatically on logout and restore it on login.

History substitutions introduce words from the history list into the input stream, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence.

History substitutions begin with the character '!'. They may begin anywhere in the input stream, but they do not nest. The '!' may be preceded by a '\' to prevent its special meaning; for convenience, a '!' is passed unchanged when it is followed by a blank, tab, newline, '=' or '('. History substitutions also occur when an input line begins with '^'. This special abbreviation will be described later. The characters used to signal history substitution ('!' and '^') can be changed by setting the *histchars* shell variable. Any input line which contains a history substitution is printed before it is executed.

A history substitution may have an ''event specification'', which indicates the event from which words are to be taken, a ''word designator'', which selects particular words from the chosen event, and / or a ''modifier'', which manipulates the selected words.

An event specification can be

n	A number, referring to a particular event
–*n*	An offset, referring to the event *n* before the current event
#	The current event. This should be used carefully in *csh*, where there is no check for recursion. *tcsh* allows 10 levels of recursion. (+)
!	The previous event (equivalent to '–1')
s	The most recent event whose first word begins with the string *s*
?*s***?**	The most recent event which contains the string **s**. The second '?' can

be omitted if it is immediately followed by a newline.

For example, consider this bit of someone's history list:

```
 9  8:30    nroff –man wumpus.man
10  8:31    cp wumpus.man wumpus.man.old
11  8:36    vi wumpus.man
12  8:37    diff wumpus.man.old wumpus.man
```

The commands are shown with their event numbers and time stamps. The current event, which we haven't typed in yet, is event 13. '!11' and '!–2' refer to event 11. '!!' refers to the previous event, 12. '!!' can be abbreviated '!' if it is followed by ':' (':' is described below). '!n' refers to event 9, which begins with 'n'. '!?old?' also refers to event 12, which contains 'old'. Without word designators or modifiers history references simply expand to the entire event, so we might type '!cp' to redo the copy command or '!! | more' if the 'diff' output scrolled off the top of the screen.

History references may be insulated from the surrounding text with braces if necessary. For example, '!vdoc' would look for a command beginning with 'vdoc', and, in this example, not find one, but '!{v}doc' would expand unambiguously to 'vi wumpus.mandoc'. Even in braces, history substitutions do not nest.

(+) While *csh* expands, for example, '!3d' to event 3 with the letter 'd' appended to it, *tcsh* expands it to the last event beginning with '3d'; only completely numeric arguments are treated as event numbers. This makes it possible to recall events beginning with numbers. To expand '!3d' as in *csh* say '!\3d'.

To select words from an event we can follow the event specification by a ':' and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

0	The first (command) word
n	The *n*th argument
^	The first argument, equivalent to '1'
$	The last argument
%	The word matched by an **?***s***?** search
x–*y*	A range of words
–*y*	Equivalent to **'0–y'**
*	Equivalent to '^–$', but returns nothing if the event contains only 1 word
*x**	Equivalent to '*x–$*'
x–	Equivalent to **'x*'**, but omitting the last word ('$')

Selected words are inserted into the command line separated by single blanks. For example, the 'diff' command in the previous example might have been typed as 'diff !!:1.old !!:1' (using ':1' to select the first argument from the previous event) or 'diff !–2:2 !–2:1' to select and swap the arguments from the 'cp' command. If we didn't care about the order of the 'diff' we might have said 'diff !–2:1–2' or simply 'diff !–2:*'. The 'cp' command might have been written 'cp wumpus.man !#:1.old', using '#' to refer to the current event. '!n:–hurkle.man' would reuse the first two words from the 'nroff' command to say 'nroff –man hurkle.man'.

The ':' separating the event specification from the word designator can be omitted if the argument selector begins with a '^', '$', '*', '%' or '–'. For example, our 'diff' command might have been 'diff !!^.old !!^' or, equivalently, 'diff !!$.old !!$'. However, if '!!' is abbreviated '!', an argument selector beginning with '–' will be interpreted as an event specification.

A history reference may have a word designator but no event specification. It then references the previous command. Continuing our 'diff' example, we could have said simply 'diff !^.old !^' or, to get the arguments in the opposite order, just 'diff !*'.

The word or words in a history reference can be edited, or ''modified'', by following it with one or more modifiers, each preceded by a ':':

h Remove a trailing pathname component, leaving the head.
t Remove all leading pathname components, leaving the tail.
r Remove a filename extension '.xxx', leaving the root name.
e Remove all but the extension.
u Uppercase the first lowercase letter.
l Lowercase the first uppercase letter.
s/*l***/***r***/**
Substitute *l* for *r*. *l* is simply a string like **r**, not a regular expression as in the *ed* command. Any character may be used as the delimiter in place of '/'; a '\' can be used to quote the delimiter inside *l* and *r*. The character '&' in the *r* is replaced by *l*; '\' also quotes '&'. If *l* is empty (""), the *l* from a previous substitution or the *s* from a previous '?*s*?' event specification is used. The trailing delimiter may be omitted if it is immediately followed by a newline.
& Repeat the previous substitution.
g Apply the following modifier once to each word.
a (+) Apply the following modifier as many times as possible to a single word. 'a' and 'g' can be used together to apply a modifier globally. In the current implementation, using the 'a' and 's' modifiers together can lead to an infinite loop. For example, ':as/f/ff/' will never terminate. This behavior might change in the future.
p Print the new command line but do not execute it.
q Quote the substituted words, preventing further substitutions.
x Like **q**, but break into words at blanks, tabs and newlines.

Modifiers are applied only to the first modifiable word (unless 'g' is used). It is an error for no word to be modifiable.

For example, the 'diff' command might have been written as 'diff wumpus.man.old !#^:r', using ':r' to remove '.old' from the first argument on the same line ('!#^'). We could say 'echo hello out there', then 'echo !*:u' to capitalize 'hello', 'echo !*:au' to say it out loud, or 'echo !*:agu' to really shout. We might follow 'mail –s "I forgot my password" rot' with '!:s/rot/root' to correct the spelling of 'root' (but see "Spelling correction" for a different approach).

There is a special abbreviation for substitutions. '^', when it is the first character on an input line, is equivalent to '!:s^'. Thus we might have said '^rot^root' to make the spelling correction in the previous example. This is the only history substitution which does not explicitly begin with '!'.

(+) In *csh* as such, only one modifier may be applied to each history or variable expansion. In *tcsh*, more than one may be used, for example

```
% mv wumpus.man /usr/man/man1/wumpus.1
% man !$:t:r
man wumpus
```

In *csh*, the result would be 'wumpus.1:r'. A substitution followed by a colon may need to be insulated from it with braces:

```
> mv a.out /usr/games/wumpus
> setenv PATH !$:h:$PATH
Bad ! modifier: $.
> setenv PATH !{-2$:h}:$PATH
setenv PATH /usr/games:/bin:/usr/bin:.
```

The first attempt would succeed in *csh* but fails in *tcsh*, because *tcsh* expects another modifier after the second colon rather than '$'.

Finally, history can be accessed through the editor as well as through the substitutions just described. The **up-** and **down-history**, **history-search-backward** and **-forward**, **i-search-back** and **-fwd**, **vi-search-back** and **-fwd**, **copy-prev-word** and **insert-last-word** editor commands search for events in the history list and copy them into the input buffer. The **toggle-literal-history** editor command switches between the expanded and literal forms of history lines in the input buffer. **expand-history** and **expand-line** expand history substitutions in the current word and in the entire input buffer respectively.

Alias substitution

The shell maintains a list of aliases which can be set, unset and printed by the **alias** and **unalias** commands. After a command line is parsed into simple commands (see "Commands") the first word of each command, left-to-right, is checked to see if it has an alias. If so, the first word is replaced by the alias. If the alias contains a history reference, it undergoes "History substitution" (q.v.) as though the original command were the previous input line. If the alias does not contain a history reference, the argument list is left untouched.

Thus if the alias for 'ls' were 'ls –l' the command 'ls /usr' would become 'ls –l /usr', the argument list here being undisturbed. If the alias for 'lookup' were 'grep !^ /etc/passwd' then 'lookup bill' would become 'grep bill /etc/passwd'. Aliases can be used to introduce parser metasyntax. For example, 'alias print 'pr \!* | lpr'' defines a ''command'' ('print') which *pr*s its arguments to the line printer.

Alias substitution is repeated until the first word of the command has no alias. If an alias substitution does not change the first word (as in the previous example) it is flagged to prevent a loop. Other loops are detected and cause an error.

Some aliases are referred to by the shell; see "Special aliases".

Variable substitution

The shell maintains a list of variables, each of which has as value a list of zero or more words. The values of shell variables can be displayed and changed with the **set** and **unset** commands. The system maintains its own list of ''environment'' variables. These can be displayed and changed with **printenv**, **setenv** and **unsetenv**.

(+) Variables may be made read-only with 'set –r' (q.v.) Read-only variables may not be modified or unset; attempting to do so will cause an error. Once made read-only, a variable cannot be made writable, so 'set –r' should be used with caution. Environment variables cannot be made read-only.

Some variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways. Some of the variables referred to by the shell are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The **–v** command line option sets this variable. "Special shell variables" lists all variables which are referred to by the shell.

Other operations treat variables numerically. The '@' command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by '$' characters. This expansion can be prevented by preceding the '$' with a '\' except within '"'s where it *always* occurs, and within ''s where it *never* occurs. Strings quoted by '' are interpreted later (see "Command substitution" below) so '$' substitution does not occur there until later, if at all. A '$' is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word (to this point) to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in '"' or given the ':q' modifier the results of variable substitution may eventually be command and filename substituted. Within '"', a variable whose value consists of multiple words expands to a (portion of a) single word, with the words of the variable's value separated by blanks. When the ':q' modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$*name*
${*name*}
Substitutes the words of the value of variable *name*, each separated by

a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter. If *name* is not a shell variable, but is set in the environment, then that value is returned (but ':' modifiers and the other forms given below are not available in this case).

$*name*[*selector*]
${*name*[*selector*]}
: Substitutes only the selected words from the value of *name*. The **selector** is subjected to '$' substitution and may consist of a single number or two numbers separated by a '–'. The first word of a variable's value is numbered '1'. If the first number of a range is omitted it defaults to '1'. If the last member of a range is omitted it defaults to '$#*name*'. The *selector* '*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$0 Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$*number*
${*number*}
: Equivalent to '$argv[*number*]'.

$* Equivalent to '$argv', which is equivalent to '$argv[*]'.

The ':' modifiers described under "History substitution", except for ':p', can be applied to the substitutions above. More than one may be used. (+) Braces may be needed to insulate a variable substitution from a literal colon just as with "History substitution" (q.v.); any modifiers must appear within the braces.

The following substitutions can not be modified with ':' modifiers.

$?*name*

${?*name*}
: Substitutes the string '1' if *name* is set, '0' if it is not.

$?0 Substitutes '1' if the current input filename is known, '0' if it is not. Always '0' in interactive shells.

$#*name*

${#*name*}
: Substitutes the number of words in *name*.

$# Equivalent to '$#argv'. (+)

$%*name*

${%*name*}
: Substitutes the number of characters in *name*. (+)

$%**number**

${%**number**}
: Substitutes the number of characters in $argv[**number**]. (+)

$? Equivalent to '$status'. (+)

$$ Substitutes the (decimal) process number of the (parent) shell.

$! Substitutes the (decimal) process number of the last background process started by this shell.

$< Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script. (+) While *csh* always quotes $<, as if it were equivalent to '$<:q', *tcsh* does not. Furthermore, when *tcsh* is waiting for a line to be typed the user may type an interrupt to interrupt the sequence into which the line is to be substituted, but *csh* does not allow this.

The editor command **expand-variables**, normally bound to '^X-$', can be used to interactively expand individual variables.

Command, filename and directory stack substitution

The remaining substitutions are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in ''. The output from such a command is broken into separate words at blanks, tabs and newlines, and null words are discarded. The output is variable and command substituted and put in place of the original string.

Command substitutions inside double quotes (") retain blanks and tabs; only newlines force new words. The single final newline does not force a new word in any case. It is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters '*', '?', '[' or '{' or begins with the character '~' it is a candidate for filename substitution, also known as ''globbing''. This word is then regarded as a pattern (''glob-pattern''), and replaced with an alphabetically sorted list of file names which match the pattern.

In matching filenames, the character '.' at the beginning of a filename or immediately following a '/', as well as the character '/' must be matched explicitly. The character '*' matches any string of characters, including the null string. The character '?' matches any single character. The sequence '[...]' matches any one of the characters enclosed. Within '[...]', a pair of characters separated by '–' matches any character lexically between the two.

(+) Some glob-patterns can be negated: The sequence '[^...]' matches any single character *not* specified by the characters and/or ranges of characters in the braces.

An entire glob-pattern can also be negated with '^':

```
> echo *
bang crash crunch ouch
> echo ^cr*
bang ouch
```

Glob-patterns which do not use '?', '*', or '[]' or which use '{}' or '~' (below) are not negated correctly.

The metanotation 'a{b,c,d}e' is a shorthand for 'abe ace ade'. Left-to-right order is preserved: '/usr/source/s1/{oldls,ls}.c' expands to '/usr/source/s1/oldls.c /usr/source/s1/ls.c'. The results of matches are sorted separately at a low level to preserve this order: '../{memo,*box}' might expand to '../memo ../box ../mbox'. (Note that 'memo' was not sorted with the results of matching '*box'.) It is not an error when this construct expands to files which do not exist, but it is possible to get an error from a command to which the expanded list is passed. This construct may be nested. As a special case the words '{', '}' and '{}' are passed undisturbed.

The character '~' at the beginning of a filename refers to home directories. Standing alone, i.e. '~', it expands to the invoker's home directory as reflected in the value of the *home* shell variable. When followed by a name consisting of letters, digits and '–' characters the shell searches for a user with that name and substitutes their home directory; thus '~ken' might expand to '/usr/ken' and '~ken/chmach' to '/usr/ken/chmach'. If the character '~' is followed by a character other than a letter or '/' or appears elsewhere than at the beginning of a word, it is left undisturbed. A command like 'setenv MANPATH /usr/man:/usr/local/man:~/lib/man' does not, therefore, do home directory substitution as one might hope.

It is an error for a glob-pattern containing '*', '?', '[' or '~', with or without '^', not to match any files. However, only one pattern in a list of glob-patterns must match a file (so that, e.g., 'rm *.a *.c *.o' would fail only if there were no files in the current directory ending in '.a', '.c', or '.o'), and if the *nonomatch* shell variable is set a pattern (or list of patterns) which matches nothing is left unchanged rather than causing an error.

The *noglob* shell variable can be set to prevent filename substitution, and the **expand-glob** editor command, normally bound to '^X-*', can be used to interactively expand individual filename substitutions.

Directory stack substitution (+)

The directory stack is a list of directories, numbered from zero, used by the **pushd**, **popd** and **dirs** builtin commands (q.v.). **dirs** can print, store in a file, restore and clear the directory stack at any time, and the *savedirs* and *dirsfile* shell variables can be set to store the directory stack automatically on logout and restore it on login. The *dirstack* shell variable can be examined to see the directory stack and set to put arbitrary directories into the directory stack.

The character '=' followed by one or more digits expands to an entry in the directory stack. The special case '=–' expands to the last directory in the stack. For example,

```
> dirs –v
0       /usr/bin
1       /usr/spool/uucp
2       /usr/accts/sys
> echo =1
/usr/spool/uucp
> echo =0/calendar
/usr/bin/calendar
> echo =–
/usr/accts/sys
```

The *noglob* and *nonomatch* shell variables and the **expand-glob** editor command apply to directory stack as well as filename substitutions.

Other substitutions (+)

There are several more transformations involving filenames, not strictly related to the above but mentioned here for completeness. *Any* filename may be expanded to a full path when the *symlinks* variable (q.v.) is set to 'expand'. Quoting prevents this expansion, and the **normalize-path** editor command does it on demand. The **normalize-command** editor command expands commands in *PATH* into full paths on demand. Finally, **cd** and **pushd** interpret '–' as the old working directory (equivalent to the shell variable *owd*). This is not a substitution at all, but an abbreviation recognized only by those commands. Nonetheless, it too can be prevented by quoting.

Commands

The next three sections describe how the shell executes commands and deals with their input and output.

Simple commands, pipelines and sequences

A simple command is a sequence of words, the first of which specifies the command to be executed. A series of simple commands joined by '|' characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next.

Simple commands and pipelines may be joined into sequences with ';', and will be executed sequentially. Commands and pipelines can also be joined into sequences with '||' or '&&', indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively.

A simple command, pipeline or sequence may be placed in parentheses, '()', to form a simple command, which may in turn be a component of a pipeline or sequence. A command, pipeline or sequence can be executed without waiting for it to terminate by following it with an '&'.

Builtin and non-builtin command execution

Builtin commands are executed within the shell. If any component of a pipeline except the last is a builtin command, the pipeline is executed in a subshell.

Parenthesized commands are always executed in a subshell.

```
(cd; pwd); pwd
```

thus prints the *home* directory, leaving you where you were (printing this after the home directory), while

```
cd; pwd
```

leaves you in the *home* directory. Parenthesized commands are most often used to prevent **cd** from affecting the current shell.

When a command to be executed is found not to be a builtin command the shell attempts to execute the command via *execve*( ). Each word in the variable *path* names a directory in which the shell will look for the command. If it is given neither a **–c** nor a **–t** option, the shell hashes the names in these directories into an internal table so that it will only try an *execve*( ) in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via **unhash**), if the shell was given a **–c** or **–t** argument or in any case for each directory component of *path* which does not begin with a '/', the shell concatenates the current working directory with the given command name to form a path name of a file which it then attempts to execute.

If the file has execute permissions but is not an executable to the system (i.e. it is neither an executable binary nor a script which specifies its interpreter), then it is assumed to be a file containing shell commands and a new shell is spawned to read it. The **shell** special alias may be set to specify an interpreter other than the shell itself.

On systems which do not understand the '#!' script interpreter convention the shell may be compiled to emulate it; see the *version* shell variable. If so, the shell checks the first line of the file to see if it is of the form '#! **interpreter** *arg* ...'. If it is, the shell starts *interpreter* with the given *arg*s and feeds the file to it on standard input.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

< *name* Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< *word* Read the shell input up to a line which is identical to *word*. *word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting '\', '"', '' or '`' appears in *word* variable and command substitution is performed on the intervening lines, allowing '\' to quote '$', '\' and '`'. Commands which are substituted have all blanks, tabs, and newlines preserved, except for

the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> *name*
>! *name*
>& *name*
>&! *name*

The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, its is truncated, its previous contents being lost.

If the shell variable *noclobber* is set, then the file must not exist or be a character special file (e.g. a terminal or '/dev/null') or an error results. This helps prevent accidental destruction of files. In this case the '!' forms can be used to suppress this check.

The forms involving '&' route the diagnostic output into the specified file as well as the standard output. *name* is expanded in the same way as '<' input filenames are.

>> name
>>& name
>>! name
>>&! *name*

Like '>', but appends output to the end of *name.* If the shell variable *noclobber* is set, then it is an error for the file *not* to exist, unless one of the '!' forms is given.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is **not** the empty file **/dev/null**, but the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see "Jobs").

Diagnostic output may be directed through a pipe with the standard output. Simply use the form '|&' rather than just '|'.

The shell cannot presently redirect diagnostic output without also redirecting standard output, but '**(***command* **>** *output-file* **) >&** *error-file*' is often an acceptable workaround. Either *output-file* or *error-file* may be '**/dev/tty**' to send output to the terminal.

Features

Having described how the shell accepts, parses and executes command lines, we now turn to a variety of its useful features.

Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The **foreach**, **switch**, and **while** statements, as well as the **if-then-else** form of the **if** statement, require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward **goto**s will succeed on non-seekable inputs.)

Expressions

The **if**, **while** and **exit** builtin commands use expressions with a common syntax. The expressions can include any of the operators described in the next three sections. Note that the **@** builtin command (q.v.) has its own separate syntax.

Logical, arithmetical and comparison operators

These operators are similar to those of C and have the same precedence. They include

```
||  &&  |  ^  &  ==  !=  =~  !~  <=  >=
<  > <<  >>  +  -  *  /  %  !  ~  (  )
```

Here the precedence increases to the right, '==' '!=' '=~' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '–', '*' '/' and '%' being, in groups, at the same level. The '==' '!=' '=~' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators '=~' and '!~' are like '!=' and '==' except that the right hand side is a glob-pattern (see "Filename substitution") against which the left hand operand is matched. This reduces the need for use of the **switch** builtin command in shell scripts when all that is really needed is pattern matching.

Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The results of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Command exit status

Commands can be executed in expressions and their exit status returned by enclosing them in braces ('{}'). Remember that the braces should be separated from the words of the command by spaces. Command executions succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the *status* shell variable examined.

File inquiry operators

Some of these operators perform true / false tests on files and related objects. They are of the form **–***op file*, where *op* is one of

r	Read access
w	Write access
x	Execute access
X	Executable in the path or shell builtin, e.g. '–X ls' and '–X ls–F' are generally true, but '–X /bin/ls' is not (+)
e	Existence
o	Ownership
z	Zero size
s	Non-zero size (+)
f	Plain file
d	Directory
l	Symbolic link (+) *
b	Block special file (+)
c	Character special file (+)
p	Named pipe (fifo) (+) *
S	Socket special file (+) *
u	Set-user-ID bit is set (+)
g	Set-group-ID bit is set (+)
k	Sticky bit is set (+)
t	*file* (which must be a digit) is an open file descriptor for a terminal device (+)
L	Applies subsequent operators in a multiple-operator test to a symbolic link rather than to the file to which the link points (+) *

file is command and filename expanded and then tested to see if it has the specified relationship to the real user. If *file* does not exist or is inaccessible or, for the operators indicated by '*', if the specified file type does not exist on the current system, then all enquiries return false, i.e. '0'.

These operators may be combined for conciseness: '**–xy file**' is equivalent to '**–x file** && **–y file**'. (+) For example, '–fx' is true (returns '1') for plain executable files, but not for directories.

L may be used in a multiple-operator test to apply subsequent operators to a symbolic link rather than to the file to which the link points. For example, '–lLo' is true for links owned by the invoking user. **Lr**, **Lw** and **Lx** are always true for links and false for non-links. **L** has a different meaning when it is the last operator in a multiple-operator test; see below.

It is possible but not useful, and sometimes misleading, to combine operators which expect *file* to be a file with operators which do not, (e.g. **X** and **t**). Following **L** with a non-file operator can lead to particularly strange results.

Other operators return other information, i.e. not just '0' or '1'. (+) They have the same format as before; *op* may be one of

A	Last file access time, as the number of seconds since the epoch
A:	Like **A**, but in timestamp format, e.g. 'Fri May 14 16:36:10 1993'
M	Last file modification time
M:	Like **M**, but in timestamp format
C	Last inode modification time
C:	Like **C**, but in timestamp format
D	Device number
I	Inode number
F	Composite **f**ile identifier, in the form **device**:**inode**
L	The name of the file pointed to by a symbolic link
N	Number of (hard) links
P	Permissions, in octal, without leading zero
P:	Like **P**, with leading zero
P*mode*	Equivalent to '–P *file* & *mode*', e.g. '–P22 *file*' returns '22' if *file* is writable by group and other, '20' if by group only, and '0' if by neither
P*mode***:**	Like **P***mode***:**, with leading zero
U	Numeric userid
U:	Username, or the numeric userid if the username is unknown
G	Numeric groupid
G:	Groupname, or the numeric groupid if the groupname is unknown
Z	Size, in bytes

Only one of these operators may appear in a multiple-operator test, and it must be the last. Note that **L** has a different meaning at the end of and elsewhere in a multiple-operator test. Because '0' is a valid return value for many of these operators, they do not return '0' when they fail: most return '–1', and **F** returns ':'.

If the shell is compiled with POSIX defined (see the *version* shell variable), the result of a file inquiry is based on the permission bits of the file and not on the result of the *access*( ) system call. For example, if one tests a file with **–w** whose permissions would ordinarily allow writing but which is on a file system mounted read-only, the test will succeed in a POSIX shell but fail in a non-POSIX shell.

File inquiry operators can also be evaluated with the **filetest** builtin command (q.v.) (+).

Jobs

The shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with '&', the shell prints a line which looks like

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the suspend key (usually ‘^Z’), which sends a STOP signal to the current job. The shell will then normally indicate that the job has been ‘Suspended’ and print another prompt. If the *listjobs* shell variable is set, all jobs will be listed like the **jobs** builtin command; if it is set to ‘long’ the listing will be in long format, like ‘jobs –l’. You can then manipulate the state of the suspended job. You can put it in the ‘‘background’’ with the **bg** command or run some other commands and eventually bring the job back into the ‘‘foreground’’ with **fg**. (See also the **run-fg-editor** editor command.) A ‘^Z’ takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. The **wait** builtin command causes the shell to wait for all background jobs to complete.

The ‘^]’ key sends a delayed suspend signal, which does not generate a STOP signal until a program attempts to **read**(2) it, to the current job. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them. The ‘^Y’ key performs this function in *csh*; in *tcsh*, ‘^Y’ is an editing command. (+)

A job being run in the background stops if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command ‘stty tostop’. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character ‘%’ introduces a job name. If you wish to refer to job number 1, you can name it as ‘%1’. Just naming a job brings it to the foreground; thus ‘%1’ is a synonym for ‘fg %1’, bringing job 1 back into the foreground. Similarly, saying ‘%1 &’ resumes job 1 in the background, just like ‘bg %1’. A job can also be named by an unambiguous prefix of the string typed in to start it: ‘%ex’ would normally restart a suspended *ex* job, if there were only one suspended job whose name began with the string ‘ex’. It is also possible to say ‘%?**string**’ to specify a job whose text contains *string*, if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a ‘+’ and the previous job with a ‘–’. The abbreviations ‘%+’, ‘%’, and (by analogy with the syntax of the **history** mechanism) ‘%%’ all refer to the current job, and ‘%–’ refers to the previous job.

The job control mechanism requires that the *stty* option ‘new’ be set on some systems. It is an artifact from a ‘new’ implementation of the tty driver which allows generation of interrupt characters from the keyboard to tell jobs to stop. See *stty* and the **setty** builtin command for details on setting options in the new tty driver.

Status reporting

The shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable *notify*, the shell will notify you immediately of changes of status in background jobs. There is also a shell command **notify** which marks a single process so that its status changes will be immediately reported. By default **notify** marks the current process; simply say 'notify' after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that 'You have stopped jobs.' You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated.

Automatic, periodic and timed events (+)

There are various ways to run commands and take other actions automatically at various times in the ''life cycle'' of the shell. They are summarized here, and described in detail under the appropriate "Builtin commands", "Special shell variables" and "Special aliases".

The **sched** builtin command puts commands in a scheduled-event list, to be executed by the shell at a given time.

The **beepcmd**, **cwdcmd**, **periodic** and **precmd** "Special aliases" can be set, respectively, to execute commands when the shell wants to ring the bell, when the working directory changes, every *tperiod* minutes and before each prompt.

The *autologout* shell variable can be set to log out or lock the shell after a given number of minutes of inactivity.

The *mail* shell variable can be set to check for new mail periodically.

The *printexitvalue* shell variable can be set to print the exit status of commands which exit with a status other than zero.

The *rmstar* shell variable can be set to ask the user, when 'rm *' is typed, if that is really what was meant.

The *time* shell variable can be set to execute the **time** builtin command after the completion of any process that takes more than a given number of CPU seconds.

The *watch* and *who* shell variables can be set to report when selected users log in or out, and the **log** builtin command reports on those users at any time.

Native Language System support (+)

The shell is eight bit clean (if so compiled; see the *version* shell variable) and thus supports character sets needing this capability. NLS support differs depending on whether or not the shell was compiled to use the system's NLS (again, see "version"). In either case, 7-bit ASCII is the default for character classification (e.g. which characters are printable) and sorting, and changing the *LANG* or *LC_CTYPE* environment variables causes a check for possible changes in these respects.

When using the system's NLS, the *setlocale*( ) function is called to determine appropriate character classification and sorting. This function typically examines the *LANG* and *LC_CTYPE* environment variables; refer to the system documentation for further details. When not using the system's NLS, the shell simulates it by assuming that the ISO 8859-1 character set is used whenever either of the *LANG* and *LC_CTYPE* variables are set, regardless of their values. Sorting is not affected for the simulated NLS.

In addition, with both real and simulated NLS, all printable characters in the range \200–\377, i.e. those that have M-**char** bindings, are automatically rebound to **self-insert-command**. The corresponding binding for the escape-**char** sequence, if any, is left alone. These characters are not rebound if the *NOREBIND* environment variable is set. This may be useful for the simulated NLS or a primitive real NLS which assumes full ISO 8859-1. Otherwise, all M-**char** bindings in the range \240–\377 are effectively undone. Explicitly rebinding the relevant keys with **bindkey** is of course still possible.

Unknown characters (i.e. those that are neither printable nor control characters) are printed in the format \nnn. If the tty is not in 8 bit mode, other 8 bit characters are printed by converting them to ASCII and using standout mode. The shell never changes the 7/8 bit mode of the tty and tracks user-initiated changes of 7/8 bit mode. NLS users (or, for that matter, those who want to use a meta key) may need to explicitly set the tty in 8 bit mode through the appropriate *stty* command in, e.g., the **~/.login** file.

OS variant support (+)

A number of new builtin commands are provided to support features in particular operating systems. All are described in detail in the "Builtin commands" section.

The *VENDOR*, *OSTYPE* and *MACHTYPE* environment variables indicate respectively the vendor, operating system and machine type (microprocessor class or machine model) of the system on which the shell thinks it is running. These are particularly useful when sharing one's home directory between several types of machines; one can, for example,

```
set path = (~/bin.$MACHTYPE /usr/ucb /bin /usr/bin .)
```

in one's **~/.login** and put executables compiled for each machine in the appropriate directory.

The *version* shell variable indicates what options were chosen when the shell was compiled.

Note also the **newgrp** builtin, the *afsuser* and *echo_style* shell variables and the system-dependent locations of the shell's input files (see "FILES").

Signal handling

Login shells ignore interrupts when reading the file **~/.logout**. The shell ignores quit signals unless started with **–q**. Login shells catch the terminate signal, but non-login shells inherit the terminate behavior from their parents. Other signals have the values which the shell inherited from its parent.

In shell scripts, the shell's handling of interrupt and terminate signals can be controlled with **onintr**, and its handling of hangups can be controlled with **hup** and **nohup**.

The shell exits on a hangup (see also the *logout* shell variable). By default, the shell's children do too, but the shell does not send them a hangup when it exits. **hup** arranges for the shell to send a hangup to a child when it exits, and **nohup** sets a child to ignore hangups.

Terminal management (+)

The shell uses three different sets of terminal (''tty'') modes: 'edit', used when editing, 'quote', used when quoting literal characters, and 'execute', used when executing commands. The shell holds some settings in each mode constant, so commands which leave the tty in a confused state do not interfere with the shell. The shell also matches changes in the speed and padding of the tty. The list of tty modes that are kept constant can be examined and modified with the **setty** builtin. Note that although the editor uses CBREAK mode (or its equivalent), it takes typed-ahead characters anyway.

The **echotc**, **settc** and **telltc** commands can be used to manipulate and debug terminal capabilities from the command line.

On systems that support SIGWINCH or SIGWINDOW, the shell adapts to window resizing automatically and adjusts the environment variables *LINES* and *COLUMNS* if set. If the environment variable *TERMCAP* contains li# and co# fields, the shell adjusts them to reflect the new window size.

REFERENCE

The next sections of this manual describe all of the available "Builtin commands", "Special aliases" and "Special shell variables".

Builtin commands

%*job*

A synonym for the **fg** builtin command.

%*job* **&**

A synonym for the **bg** builtin command.

: Does nothing, successfully.

@ Print the values of all shell variables.

@ *name* **=** *expr*

Set the specified *name* to the value of *expr*. If *expr* contains '<', '>', '&' or '' then at least that part of *expr* must be placed within '()'.

@ *name*[*index*] **=** *expr*

Assign the value of *expr* to the *index*'th argument of *name*. Both *name* and its *index*'th component must already exist.

expr may contain the operators '*', '+', etc. as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix '++' and '—' operators increment and decrement *name* respectively, e.g. '@ i++'.

Note that the syntax of *expr* has nothing to do with that described under "Expressions".

alias [*name* [*wordlist*]]

Without arguments, prints all aliases. With *name*, prints the alias for name. With *name* and *wordlist*, assigns *wordlist* as the alias of *name*. *wordlist* is command and filename substituted. *name* may not be 'alias' or 'unalias'. See also the **unalias** builtin command.

alloc

Shows the amount of dynamic memory acquired, broken down into used and free memory. With an argument shows the number of free and used blocks in each size category. The categories start at size 8 and double at each step. This command's output may vary across system types, since systems other than the VAX may use a different memory allocator.

bg [**%***job* ...]

Puts the specified jobs (or, without arguments, the current job) into the background, continuing each if it is stopped. **job** may be a number, a string, '', '%', '+' or '–' as described under "Jobs".

bindkey [**–l** | **–d** | **–e** | **–v** | **–u**] (+)

bindkey [**–a**] [**–b**] [**–k**] [**–r**] [**—**] *key* (+)

bindkey [**–a**] [**–b**] [**–k**] [**–c** | **–s**] [**—**] *key command* (+)

Without options, the first form lists all bound keys and the editor command to which each is bound, the second form lists the editor command to which *key* is bound and the third form binds the editor command *command* to *key*. Options include:

- **–l** Lists all editor commands and a short description of each.
- **–d** Binds all keys to the standard bindings for the default editor.
- **–e** Binds all keys to the standard GNU Emacs-like bindings.
- **–v** Binds all keys to the standard *vi*-like bindings.
- **–a** Lists or changes key-bindings in the alternative key map. This is the key map used in *vi* command mode.
- **–b** *key* is interpreted as a control character written ^*character* (e.g. '^A') or C-*character* (e.g. 'C-A'), a meta character written M-*character* (e.g. 'M-A'), a function key written F-*string* (e.g. 'F-string'), or an extended prefix key written X-*character* (e.g. 'X-A').
- **–k** *key* is interpreted as a symbolic arrow key name, which may be one of 'down', 'up', 'left' or 'right'.
- **–r** Removes *key*'s binding. Be careful: 'bindkey –r' does *not* bind *key* to **self-insert-command** (q.v.), it unbinds *key* completely.
- **–c** *command* is interpreted as a builtin or external command

instead of an editor command.

–s *command* is taken as a literal string and treated as terminal input when *key* is typed. Bound keys in *command* are themselves reinterpreted, and this continues for ten levels of interpretation.

— Forces a break from option processing, so the next word is taken as *key* even if it begins with '–'.

–u (or any invalid option)
Prints a usage message.

key may be a single character or a string. If a command is bound to a string, the first character of the string is bound to **sequence-lead-in** and the entire string is bound to the command.

Control characters in *key* can be literal (they can be typed by preceding them with the editor command **quoted-insert**, normally bound to '^V') or written caret-character style, e.g. '^A'. Delete is written '^?' (caret-question mark). *key* and *command* can contain backslashed escape sequences (in the style of System V *echo*) as follows:

\a Bell
\b Backspace
\e Escape
\f Form feed
\n Newline
\r Carriage return
\t Horizontal tab
\v Vertical tab
*****nnn* The ASCII character corresponding to the octal number *nnn*

'\' nullifies the special meaning of the following character, if it has any, notably '\' and '^'.

break
Causes execution to resume after the **end** of the nearest enclosing **foreach** or **while**. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a **switch**, resuming after the **endsw**.

builtins (+)
Prints the names of all builtin commands.

bye (+)
A synonym for the **logout** builtin command. Available only if the shell was so compiled; see the *version* shell variable.

case *label***:**
A label in a **switch** statement as discussed below.

cd [**–p**] [**–l**] [**–n** | **–v**] [*name*]
If a directory *name* is given, changes the shell's working directory to *name*. If not, changes to *home*. If *name* is '–' it is interpreted as the previous working directory (see "Other substitutions"). (+) If *name* is

not a subdirectory of the current directory (and does not begin with '/', './' or '../'), each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

With **–p**, prints the final directory stack, just like **dirs**. The **–l**, **–n** and **–v** flags have the same effect on **cd** as on **dirs**, and they imply **–p**. (+)

chdir

A synonym for the **cd** builtin command.

complete [*command* [*word*/*pattern*/*list*[**:***select*]/[[*suffix*]/] ...]] (+)

Without arguments, lists all completions. With *command*, lists completions for *command*. With *command* and *word* etc., defines completions.

command may be a full command name or a glob-pattern (see "Filename substitution"). It can begin with '–' to indicate that completion should be used only when *command* is ambiguous.

word specifies which word relative to the current word is to be completed, and may be one of the following:

c Current-word completion. *pattern* is a glob-pattern which must match the beginning of the current word on the command line. *pattern* is ignored when completing the current word.

C Like **c**, but includes *pattern* when completing the current word.

n Next-word completion. *pattern* is a glob-pattern which must match the beginning of the previous word on the command line.

N Like **n**, but must match the beginning of the word two before the current word.

p Position-dependent completion. *pattern* is a numeric range, with the same syntax used to index shell variables, which must include the current word.

list, the list of possible completions, may be one of the following:

a Aliases

b Bindings (editor commands)

c Commands (builtin or external commands)

C External commands which begin with the supplied path prefix

d Directories

D Directories which begin with the supplied path prefix

e Environment variables

f Filenames

F Filenames which begin with the supplied path prefix

g Groupnames

j Jobs

l Limits

n Nothing

s	Shell variables
S	Signals
t	Plain (‘‘text’’) files
T	Plain (‘‘text’’) files which begin with the supplied path prefix
v	Any variables
u	Usernames
x	Like **n**, but prints *select* when **list-choices** is used.
X	Completions
$*var*	Words from the variable *var*
(...)	Words from the given list
‘...‘	Words from the output of command

select is an optional glob-pattern. If given, only words from *list* which match *select* are considered and the *fignore* shell variable is ignored. The last three types of completion may not have a *select* pattern, and *x* uses *select* as an explanatory message when the **list-choices** editor command is used.

suffix is a single character to be appended to a successful completion. If null, no character is appended. If omitted (in which case the fourth delimiter can also be omitted), a slash is appended to directories and a space to other words.

Now for some examples. Some commands take only directories as arguments, so there’s no point completing plain files.

```
> complete cd 'p/1/d/'
```

completes only the first word following ‘cd’ (‘p/1’) with a directory. **p**-type completion can also be used to narrow down command completion:

```
> co[^D]
complete compress
> complete –co* 'p/0/(compress)/'
> co[^D]
> compress
```

This completion completes commands (words in position 0, ‘p/0’) which begin with ‘co’ (thus matching ‘co*’) to ‘compress’ (the only word in the list). The leading ‘–’ indicates that this completion is to be used only with ambiguous commands.

```
> complete find 'n/–user/u/'
```

is an example of **n**-type completion. Any word following ‘find’ and immediately following ‘–user’ is completed from the list of users.

```
> complete cc 'c/–I/d/'
```

demonstrates **c**-type completion. Any word following ‘cc’ and beginning with ‘–I’ is completed as a directory. ‘–I’ is not taken as part of the directory because we used lowercase **c**.

Different *list*s are useful with different commands.

```
> complete alias 'p/1/a/'
> complete man 'p/*/c/'
> complete set 'p/1/s/'
> complete true 'p/1/x:Truth has no options./'
```

These complete words following 'alias' with aliases, 'man' with commands, and 'set' with shell variables. 'true' doesn't have any options, so **x** does nothing when completion is attempted and prints 'Truth has no options.' when completion choices are listed.

Note that the *man* example, and several other examples below, could just as well have used 'c/*' or 'n/*' as 'p/*'.

Words can be completed from a variable evaluated at completion time,

```
> complete ftp 'p/1/$hostnames/'
> set hostnames = (rtfm.mit.edu tesla.ee.cornell.edu)
> ftp [^D]
rtfm.mit.edu tesla.ee.cornell.edu
> ftp [^C]
> set hostnames = (rtfm.mit.edu tesla.ee.cornell.edu uunet.uu.net)
> ftp [^D]
rtfm.mit.edu tesla.ee.cornell.edu uunet.uu.net
```

or from a command run at completion time:

```
> complete kill 'p/*/`ps | awk \{print\ \$1\}`/'
> kill -9 [^D]
23113 23377 23380 23406 23429 23529 23530 PID
```

Note that the **complete** command does not itself quote its arguments, so the braces, space and '$' in '{print $1}' must be quoted explicitly.

One command can have multiple completions:

```
> complete dbx 'p/2/(core)/' 'p/*/c/'
```

completes the second argument to 'dbx' with the word 'core' and all other arguments with commands. Note that the positional completion is specified before the next-word completion. Since completions are evaluated from left to right, if the next-word completion were specified first it would always match and the positional completion would never be executed. This is a common mistake when defining a completion.

The *select* pattern is useful when a command takes only files with particular forms as arguments. For example,

```
> complete cc 'p/*/f:*.[cao]/'
```

completes 'cc' arguments only to files ending in '.c', '.a', or '.o'. **select** can also exclude files, using negation of a glob-pattern as described under "Filename substitution". One might use

```
> complete rm 'p/*/f:^*.{c,h,cc,C,tex,1,man,l,y}/'
```

to exclude precious source code from 'rm' completion. Of course, one could still type excluded names manually or override the completion

mechanism using the **complete-word-raw** or **list-choices-raw** editor commands (q.v.).

The 'C', 'D', 'F' and 'T' **list**s are like 'c', 'd', 'f' and 't' respectively, but they use the *select* argument in a different way: to restrict completion to files beginning with a particular path prefix. For example, the Elm mail program uses '=' as an abbreviation for one's mail directory. One might use

```
> complete elm 'c@=@F:$HOME/Mail/@'
```

to complete 'elm –f =' as if it were 'elm –f ~/Mail/'. Note that we used '@' instead of '/' to avoid confusion with the *select* argument, and we used '$HOME' instead of '~' because home directory substitution only works at the beginning of a word.

suffix is used to add a nonstandard suffix (not space or '/' for directories) to completed words.

```
> complete finger 'c/*@/$hostnames/' 'p/1/u/@'
```

completes arguments to 'finger' from the list of users, appends an '@', and then completes after the '@' from the 'hostnames' variable. Note again the order in which the completions are specified.

Finally, here's a complex example for inspiration:

```
> complete find \
´n/–name/f/' 'n/–newer/f/' 'n/–{,n}cpio/f/' \
´n/–exec/c/' 'n/–ok/c/' 'n/–user/u/' \
´n/–group/g/' 'n/–fstype/(nfs 4.2)/' \
´n/–type/(b c d f l p s)/' \
´c/–/(name newer cpio ncpio exec ok user \
group fstype type atime ctime depth inum \
ls mtime nogroup nouser perm print prune \
size xdev)/' \
´p/*/d/'
```

This completes words following '–name', '–newer', '–cpio' or 'ncpio' (note the pattern which matches both) to files, words following '–exec' or '–ok' to commands, words following 'user' and 'group' to users and groups respectively and words following '–fstype' or '–type' to members of the given lists. It also completes the switches themselves from the given list (note the use of **c**-type completion) and completes anything not otherwise completed to a directory. Whew.

Remember that programmed completions are ignored if the word being completed is a tilde substitution (beginning with '~') or a variable (beginning with '$'). **complete** is an experimental feature, and the syntax may change in future versions of the shell. See also the **uncomplete** builtin command.

continue

Continues execution of the nearest enclosing **while** or **foreach**. The rest of the commands on the current line are executed.

default:

Labels the default case in a **switch** statement. It should come after all **case** labels.

dirs [**–l**] [**–n** | **–v**]

dirs –S | **–L** [*filename*] (+)

dirs –c (+)

The first form prints the directory stack. The top of the stack is at the left and the first directory in the stack is the current directory. With **–l**, '~' or '~*name*' in the output is expanded explicitly to *home* or the pathname of the home directory for user *name*. (+) With **–n**, entries are wrapped before they reach the edge of the screen. (+) With **–v**, entries are printed one per line, preceded by their stack positions. (+) If more than one of **–n** or **–v** is given, **–v** takes precedence. **–p** is accepted but does nothing.

With **–S**, the second form saves the directory stack to **filename** as a series of **cd** and **pushd** commands. With **–L**, the shell sources *filename*, which is presumably a directory stack file saved by the **–S** option or the *savedirs* mechanism. In either case, *dirsfile* is used if *filename* is not given and **~/.cshdirs** is used if *dirsfile* is unset.

Note that login shells do the equivalent of 'dirs –L' on startup and, if *savedirs* is set, 'dirs –S' before exiting. Because only **~/.tcshrc** is normally sourced before **~/.cshdirs**, *dirsfile* should be set in **~/.tcshrc** rather than **~/.login**.

The last form clears the directory stack.

echo [**–n**] *word* ...

Writes each *word* to the shell's standard output, separated by spaces and terminated with a newline. The *echo_style* shell variable may be set to emulate (or not) the flags and escape sequences of the BSD and / or System V versions of *echo*; see *echo*.

echotc [**–sv**] *arg* ... (+)

Exercises the terminal capabilities (see *termcap*) in *args*. For example, 'echotc home' sends the cursor to the home position, 'echotc cm 3 10' sends it to column 3 and row 10, and 'echotc ts 0; echo "This is a test."; echotc fs' prints "This is a test." in the status line.

If *arg* is 'baud', 'cols', 'lines', 'meta' or 'tabs', prints the value of that capability ("yes" or "no" indicating that the terminal does or does not have that capability). One might use this to make the output from a shell script less verbose on slow terminals, or limit command output to the number of lines on the screen:

```
> set history='echotc lines'
> @ history--
```

Termcap strings may contain wildcards which will not echo correctly. One should use double quotes when setting a shell variable to a terminal capability string, as in the following example that places the date in the status line:

```
> set tosl="‘echotc ts 0‘"
> set frsl="‘echotc fs‘"
> echo –n "$tosl";date; echo –n "$frsl"
```

With **–s**, nonexistent capabilities return the empty string rather than causing an error. With **–v**, messages are verbose.

else

end

endif

endsw

See the description of the **foreach**, **if**, **switch**, and **while** statements below.

eval *arg ...*

Treats the arguments as input to the shell and executes the resulting command(s) in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset* for a sample use of **eval**.

exec *command*

Executes the specified command in place of the current shell.

exit [*expr*]

The shell exits either with the value of the specified *expr* (an expression, as described under “Expressions”) or, without *expr*, with the value of the *status* variable.

fg [**%***job ...*]

Brings the specified jobs (or, without arguments, the current job) into the foreground, continuing each if it is stopped. *job* may be a number, a string, ‘’, ‘%’, ‘+’ or ‘–’ as described under “Jobs”. See also the **run-fg-editor** editor command.

filetest –*op file ...* (+)

Applies *op* (which is a file inquiry operator as described under “File inquiry operators”) to each **file** and returns the results as a space-separated list.

foreach *name* **(***wordlist***)**

...

end

Successively sets the variable *name* to each member of *wordlist* and executes the sequence of commands between this command and the matching **end**. (Both **foreach** and **end** must appear alone on separate lines.) The builtin command **continue** may be used to continue the loop prematurely and the builtin command **break** to terminate it prematurely. When this command is read from the terminal, the loop is read once prompting with ‘foreach? ’ (or *prompt2*) before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

getspath (+)
Prints the system execution path. (TCF only)

getxvers (+)
Prints the experimental version prefix. (TCF only)

glob *wordlist*
Like **echo**, but no '\' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto *word*
word is filename and command-substituted to yield a string of the form 'label'. The shell rewinds its input as much as possible, searches for a line of the form 'label:', possibly preceded by blanks or tabs, and continues execution after that line.

hashstat
Prints a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding **exec**'s). An **exec** is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component which does not begin with a '/'.

On machines without *vfork*, prints only the number and size of hash buckets.

history [**–hr**] [*n*]

history –S | **–L** | **–M** [*filename*] (+)

history –c (+)
The first form prints the history event list. If *n* is given only the *n* most recent events are printed or saved. With **–h**, the history list is printed without leading numbers and with time-stamps in comment form. (This can be used to produce files suitable for loading with 'history –L' or 'source –h'.) With **–r**, the order of printing is most recent first rather than oldest first.

With **–S**, the second form saves the history list to *filename*. If the first word of the *savehist* shell variable is set to a number, at most that many lines are saved. If the second word of *savehist* is set to 'merge', the history list is merged with the existing history file instead of replacing it (if there is one) and sorted by time stamp. (+) Merging is intended for an environment like the X Window System with several shells in simultaneous use. Currently it only succeeds when the shells quit nicely one after another.

With **–L**, the shell appends *filename*, which is presumably a history list saved by the **–S** option or the *savehist* mechanism, to the history list. **–M** is like **–L**, but the contents of *filename* are merged into the history list and sorted by timestamp. In either case, *histfile* is used if *filename* is not given and **~/.history** is used if *histfile* is unset. 'history –L' is exactly like 'source –h' except that it does not require a filename.

Note that login shells do the equivalent of 'history –L' on startup and, if *savehist* is set, 'history –S' before exiting. Because only **~/.tcshrc** is

normally sourced before **~/.history**, *histfile* should be set in **~/.tcshrc** rather than **~/.login**.

If *histlit* is set, the first and second forms print and save the literal (unexpanded) form of the history list.

The last form clears the history list.

hup [*command*] (+)

With *command*, runs *command* such that it will exit on a hangup signal and arranges for the shell to send it a hangup signal when the shell exits. Note that commands may set their own response to hangups, overriding **hup**. Without an argument (allowed only in a shell script), causes the shell to exit on a hangup for the remainder of the script. See also "Signal handling" and the **nohup** builtin command.

if (*expr***)** *command*

If *expr* (an expression, as described under "Expressions") evaluates true, then *command* is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the **if** command. *command* must be a simple command, not an alias, a pipeline, a command list or a parenthesized command list, but it may have arguments. Input/output redirection occurs even if *expr* is false and *command* is thus *not* executed; this is a bug.

if (*expr***) then**

...

else if (*expr2***) then**

...

else

...

endif

If the specified *expr* is true then the commands to the first **else** are executed; otherwise if *expr2* is true then the commands to the second **else** are executed, etc. Any number of **else-if** pairs are possible; only one **endif** is needed. The **else** part is likewise optional. (The words **else** and **endif** must appear at the beginning of input lines; the **if** must appear alone on its input line or after an **else**.)

inlib *shared-library* **... (+)**

Adds each *shared-library* to the current environment. There is no way to remove a shared library. (Domain/OS only)

jobs [**–l**]

Lists the active jobs. With **–l**, lists process IDs in addition to the normal information. On TCF systems, prints the site on which each job is executing.

kill [**–***signal*] **%***job* | *pid* ...

kill –l

The first form sends the specified *signal* (or, if none is given, the TERM (terminate) signal) to the specified jobs or processes. *job* may be a

number, a string, ‘’, ‘%’, ‘+’ or ‘–’ as described under “Jobs”. Signals are either given by number or by name (as given in **$OPENNT_ROOT/usr/include/signal.h**, stripped of the prefix ‘SIG’). There is no default *job*; saying just ‘kill’ does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well. The second form lists the signal names.

limit [**–h**] [*resource* [*maximum-use*]]

Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource*. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given. If the **–h** flag is given, the hard limits are used instead of the current limits. The hard limits impose a ceiling on the values of the current limits. Only the super-user may raise the hard limits, but a user may lower or raise the current limits within the legal range.

Controllable resources currently include **cputime** (the maximum number of cpu-seconds to be used by each process), **filesize** (the largest single file which can be created), **datasize** (the maximum growth of the data+stack region via *sbrk*( ) beyond the end of the program text), **stacksize** (the maximum size of the automatically-extended stack region), **coredumpsize** (the size of the largest core dump that will be created), and **memoryuse**, the maximum amount of physical memory a process may have allocated to it at a given time.

maximum-use may be given as a (floating point or integer) number followed by a scale factor. For all limits other than **cputime** the default scale is ‘k’ or ‘kilobytes’ (1024 bytes); a scale factor of ‘m’ or ‘megabytes’ may also be used. For **cputime** the default scaling is ‘seconds’, while ‘m’ for minutes or ‘h’ for hours, or a time of the form ‘mm:ss’ giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

log (+)

Prints the *watch* shell variable and reports on each user indicated in *watch* who is logged in, regardless of when they last logged in. See also *watchlog*.

login

Terminates a login shell, replacing it with an instance of **$OPENNT_ROOT/bin/login**. This is one way to log off, included for compatibility with *sh*.

logout

Terminates a login shell. Especially useful if *ignoreeof* is set.

ls–F [*–switch* ...] [*file* ...] (+)

Lists files like ‘ls –F’, but much faster. It identifies each type of special file in the listing with a special character:

/	Directory
*	Executable

#	Block device
%	Character device
|	Named pipe (systems with named pipes only)
=	Socket (systems with sockets only)
@	Symbolic link (systems with symbolic links only)
+	Hidden directory (AIX only) or context dependent (HP/UX only)
:	Network special (HP/UX only)

If the *listlinks* shell variable is set, symbolic links are identified in more detail (only, of course, on systems which have them):

@	Symbolic link to a non-directory
>	Symbolic link to a directory
&	Symbolic link to nowhere

listlinks also slows down **ls–F** and causes partitions holding files pointed to by symbolic links to be mounted.

If the *listflags* shell variable is set to 'x', 'a' or 'A', or any combination thereof (e.g. 'xA'), they are used as flags to **ls–F**, making it act like 'ls –xF', 'ls –Fa', 'ls –FA' or a combination (e.g. 'ls –FxA'). On machines where 'ls –C' is not the default, **ls–F** acts like 'ls –CF', unless *listflags* contains an 'x', in which case it acts like 'ls –xF'. **ls–F** passes its arguments to *ls* if it is given any switches, so 'alias ls ls–F' generally does the right thing. **ls–F** includes file identification characters when sorting filenames; this is a bug.

migrate [**–***site*] *pid* | **%***jobid* ... (+)

migrate –*site* (+)

The first form migrates the process or job to the site specified or the default site determined by the system path. The second form is equivalent to 'migrate –**site** $$': it migrates the current process to the specified site. Migrating the shell itself can cause unexpected behavior, since the shell does not like to lose its tty. (TCF only)

newgrp [**–**] *group* (+)

Equivalent to 'exec newgrp'; see *newgrp*. Available only if the shell was so compiled; see the *version* shell variable.

nice [**+***number*] [*command*]

Sets the scheduling priority for the shell to *number*, or, without *number*, to 4. With *command*, runs *command* at the appropriate priority. The greater the *number*, the less cpu the process gets. The super-user may specify negative priority by using 'nice –number ...'. Command is always executed in a sub-shell, and the restrictions placed on commands in simple **if** statements apply.

nohup [*command*]

With *command*, runs *command* such that it will ignore hangup signals. Note that commands may set their own response to hangups, overriding **nohup**. Without an argument (allowed only in a shell script), causes the shell to ignore hangups for the remainder of the script. See also "Signal handling" and the **hup** builtin command.

notify [**%***job* ...]

Causes the shell to notify the user asynchronously when the status of any of the specified jobs (or, without %*job*, the current job) changes, instead of waiting until the next prompt as is usual. *job* may be a number, a string, '', '%', '+' or '–' as described under "Jobs9cd. See also the *notify* shell variable.

onintr [**–** | *label*]

Controls the action of the shell on interrupts. Without arguments, restores the default action of the shell on interrupts, which is to terminate shell scripts or to return to the terminal command input level. With '–', causes all interrupts to be ignored. With *label*, causes the shell to execute a 'goto **label**' when an interrupt is received or a child process terminates because it was interrupted.

onintr is ignored if the shell is running detached and in system startup files (see "FILES"), where interrupts are disabled anyway.

popd [**–p**] [**–l**] [**–n** | **–v**] [**+***n*]

Without arguments, pops the directory stack and returns to the new top directory. With a number '+*n*', discards the *n*'th entry in the stack.

Finally, all forms of **popd** print the final directory stack, just like **dirs**. The *pushdsilent* shell variable can be set to prevent this and the **–p** flag can be given to override *pushdsilent*. The **–l**, **–n** and **–v** flags have the same effect on **popd** as on **dirs**. (+)

printenv [*name*] (+)

Prints the names and values of all environment variables or, with *name*, the value of the environment variable *name*.

pushd [**–p**] [**–l**] [**–n** | **–v**] [*name* | **+***n*]

Without arguments, exchanges the top two elements of the directory stack. If *pushdtohome* is set, **pushd** without arguments does 'pushd ~', like **cd**. (+) With *name*, pushes the current working directory onto the directory stack and changes to *name*. If *name* is '–' it is interpreted as the previous working directory (see "Filename substitution"). (+) If *dunique* is set, **pushd** removes any instances of *name* from the stack before pushing it onto the stack. (+) With a number '+*n*', rotates the *n*th element of the directory stack around to be the top element and changes to it. If *dextract* is set, however, 'pushd +*n*' extracts the *n*th directory, pushes it onto the top of the stack and changes to it. (+)

Finally, all forms of **pushd** print the final directory stack, just like **dirs**. The *pushdsilent* shell variable can be set to prevent this and the **–p** flag can be given to override *pushdsilent*. The **–l**, **–n** and **–v** flags have the same effect on **pushd** as on **dirs**. (+)

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories. Also flushes the cache of home directories built by tilde

expansion.

repeat *count command*
The specified *command*, which is subject to the same restrictions as the *command* in the one line **if** statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

rootnode *//nodename* (+)
Changes the rootnode to *//nodename, so that '/' will be interpreted as '//nodename*'. (Domain/OS only)

sched (+)

sched [**+**]*hh:mm command* (+)

sched –n (+)
The first form prints the scheduled-event list. The *sched* shell variable may be set to define the format in which the scheduled-event list is printed. The second form adds *command* to the scheduled-event list. For example,

```
> sched 11:00 echo It\'s eleven o\'clock.
```

causes the shell to echo 'It's eleven o'clock.' at 11 AM. The time may be in 12-hour AM/PM format

```
> sched 5pm set prompt='[%h] It\'s after 5; go home: >'
```

or may be relative to the current time:

```
> sched +2:15 /usr/lib/uucp/uucico –r1 –sother
```

A relative time specification may not use AM/PM format. The third form removes item *n* from the event list:

```
> sched
     1  Wed Apr  4 15:42  /usr/lib/uucp/uucico –r1 –sother
     2  Wed Apr  4 17:00  set prompt=[%h] It's after 5; go home: >
> sched –2
> sched
     1  Wed Apr  4 15:42  /usr/lib/uucp/uucico –r1 –sother
```

A command in the scheduled-event list is executed just before the first prompt is printed after the time when the command is scheduled. It is possible to miss the exact time when the command is to be run, but an overdue command will execute at the next prompt. A command which comes due while the shell is waiting for user input is executed immediately. However, normal operation of an already-running command will not be interrupted so that a scheduled-event list element may be run.

This mechanism is similar to, but not the same as, the *at* command on some traditional systems. Its major disadvantage is that it may not run a command at exactly the specified time. Its major advantage is that because **sched** runs directly from the shell, it has access to shell variables and other structures. This provides a mechanism for changing one's working environment based on the time of day.

set

set *name* ...

set *name*=*word* ...

set *name*=**(***wordlist***)** ...

set *name***[***index***]**=*word* ...

set –r (+)

set –r *name* ... *(+)*

set –r *name*=*word* ... (+)

set –r *name*=**(***wordlist***)** ... (+)

The first form of the command prints the value of all shell variables. Variables which contain more than a single word print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*.

The fourth form sets *name* to the list of words in **wordlist**. In all cases the value is command and filename expanded. The fifth form sets the **index**'th component of name to **word**; this component must already exist. The sixth form lists the names (only) of all shell variables which are read-only. The seventh form makes *name* read-only, whether or not it has a value. The second form sets *name* to the null string. The eighth and ninth forms are the same as the third and fourth forms, but make *name* read-only at the same time.

These arguments can be repeated to set and / or make read-only multiple variables in a single set command. Note, however, that variable expansion happens for all arguments before any setting occurs. Note also that '=' can be adjacent to both *name* and *word* or separated from both by whitespace, but cannot be adjacent to only one or the other. See also the **unset** builtin command.

setenv [*name* [*value*]]
Without arguments, prints the names and values of all environment variables. Given *name*, sets the environment variable *name* to *value* or, without *value*, to the null string.

setpath *path* (+)
Equivalent to *setpath*. (Mach only)

setspath LOCAL | *site* | *cpu* ... (+)
Sets the system execution path. (TCF only)

settc *cap value* (+)
Tells the shell to believe that the terminal capability *cap* (as defined in **termcap**(5)) has the value *value*. No sanity checking is done. Concept terminal users may have to 'settc xn no' to get proper wrapping at the rightmost column.

setty [**–d** | **–q** | **–x**] [**–a**] [[**+** | **–**]*mode*] (+)
Controls which tty modes (see "Terminal management") the shell does not allow to change. **–d**, **–q** or **–x** tells **setty** to act on the 'edit', 'quote' or 'execute' set of tty modes respectively; without **–d**, **–q** or **–x**,

'execute' is used.

Without other arguments, **setty** lists the modes in the chosen set which are fixed on ('+mode') or off ('–mode'). The available modes, and thus the display, vary from system to system. With **–a**, lists all tty modes in the chosen set whether or not they are fixed. With **+***mode*, **–***mode* or *mode*, fixes *mode* on or off or removes control from *mode* in the chosen set. For example, 'setty +echok echoe' fixes 'echok' mode on and allows commands to turn 'echoe' mode on or off, both when the shell is executing commands.

setxvers [*string*] (+)
Set the experimental version prefix to *string*, or removes it if *string* is omitted. (TCF only)

shift [*variable*]
Without arguments, discards *argv*[1] and shifts the members of *argv* to the left. It is an error for *argv* not to be set or to have less than one word as value. With *variable*, performs the same function on *variable*.

source [**–h**] *name* [**args** ...]
The shell reads and executes commands from *name*. The commands are not placed on the history list. If any *args* are given, they are placed in **argv**. (+) **source** commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a **source** at any level terminates all nested **source** commands. With **–h**, commands are placed on the history list instead of being executed, much like 'history –L'.

stop %*job* | *pid* ...
Stops the specified jobs or processes which are executing in the background. *job* may be a number, a string, '', '%', '+' or '–' as described under "Jobs". There is no default *job*; saying just 'stop' does not stop the current job.

suspend
Causes the shell to stop in its tracks, much as if it had been sent a stop signal with **^Z**. This is most often used to stop shells started by *su*.

switch (*string***)**

case *str1***:**

...

breaksw

...

default:

...

breaksw

endsw
Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters '*', '?' and '[...]' may be used in the case labels, which

are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command **breaksw** causes execution to continue after the **endsw**. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the **endsw**.

telltc (+)
Lists the values of all terminal capabilities (see **termcap**(5)).

time [*command*]
Executes *command* (which must be a simple command, not an alias, a pipeline, a command list or a parenthesized command list) and prints a time summary as described under the *time* variable. If necessary, an extra shell is created to print the time statistic when the command completes. Without *command*, prints a time summary for the current shell and its children.

umask [*value*]
Sets the file creation mask to *value*, which is given in octal. Common values for the mask are 002, giving all access to the group and read and execute access to others, and 022, giving read and execute access to the group and others. Without *value*, prints the current file creation mask.

unalias *pattern*
Removes all aliases whose names match *pattern*. 'unalias *' thus removes all aliases. It is not an error for nothing to be **unalias**ed.

uncomplete *pattern* **(+)**
Removes all completions whose names match *pattern*. 'uncomplete *' thus removes all completions. It is not an error for nothing to be **uncomplete**d.

unhash
Disables use of the internal hash table to speed location of executed programs.

universe *universe (+)*
Sets the universe to **universe**. (Masscomp/RTU only)

unlimit [**–h**] [*resource*]
Removes the limitation on *resource* or, if no *resource* is specified, all *resource* limitations. With **–h**, the corresponding hard limits are removed. Only the super-user may do this.

unset *pattern*
Removes all variables whose names match *pattern*, unless they are read-only. 'unset *' thus removes all variables unless they are read-only; this is a bad idea. It is not an error for nothing to be **unset**.

unsetenv *pattern*
Removes all environment variables whose names match *pattern*. 'unsetenv *' thus removes all environment variables; this is a bad idea. It is not an error for nothing to be **unsetenv**ed.

ver [*systype* [*command*]] (+)
Without arguments, prints *SYSTYPE*. With *systype*, sets *SYSTYPE* to *systype*. With *systype* and *command*, executes *command* under *systype*. *systype* may be 'bsd4.3' or 'sys5.3'. (Domain/OS only)

wait
The shell waits for all background jobs. If the shell is interactive, an interrupt will disrupt the wait and cause the shell to print the names and job numbers of all outstanding jobs.

warp *universe* (+)
Sets the universe to *universe*. (Convex/OS only)

watchlog (+)
An alternate name for the **log** builtin command (q.v.). Available only if the shell was so compiled; see the *version* shell variable.

where *command* (+)
Reports all known instances of *command*, including aliases, builtins and executables in *path*.

which *command* (+)
Displays the command that will be executed by the shell after substitutions, *path* searching, etc. The builtin command is just like *which*, but it correctly reports *tcsh* aliases and builtins and is 10 to 100 times faster. See also the **which-command** editor command.

while (*expr***)**

...

end Executes the commands between the **while** and the matching **end** while *expr* (an expression, as described under "Expressions") evaluates non-zero. **while** and **end** must appear alone on their input lines. **break** and **continue** may be used to terminate or continue the loop prematurely. If the input is a terminal, the user is prompted the first time through the loop as with **foreach**.

Special aliases (+)

If set, each of these aliases executes automatically at the indicated time. They are all initially undefined.

beepcmd
Runs when the shell wants to ring the terminal bell.

cwdcmd
Runs after every change of working directory. For example, if the user is working on an X window system using *xterm* and a re-parenting window manager that supports title bars such as *twm* and does

```
> alias cwdcmd  'echo -n "^[]2;${HOST}:$cwd ^G"'
```

then the shell will change the title of the running *xterm* to be the name of the host, a colon, and the full current working directory. A fancier way to do that is

```
> alias cwdcmd 'echo -n "^[]2;${HOST}:$cwd^G^[]1;${HOST}^G"'
```

This will put the hostname and working directory on the title bar but only the hostname in the icon manager menu.

Note that putting a **cd**, **pushd** or **popd** in **cwdcmd** may cause an infinite loop. It is the author's opinion that anyone doing so will get what they deserve.

periodic

Runs every *tperiod* minutes. This provides a convenient means for checking on common but infrequent changes such as new mail. For example, if one does

```
> set tperiod = 30
> alias periodic checknews
```

then the *checknews* program runs every 30 minutes. If *periodic* is set but *tperiod* is unset or set to 0, **periodic** behaves like **precmd**.

precmd

Runs just before each prompt is printed. For example, if one does

```
> alias precmd date
```

then *date* runs just before the shell prompts for each command. There are no limits on what **precmd** can be set to do, but discretion should be used.

shell

Specifies the interpreter for executable scripts which do not themselves specify an interpreter. The first word should be a full path name to the desired interpreter (e.g. '/bin/csh' or '/usr/local/bin/tcsh').

Special shell variables

The variables described in this section have special meaning to the shell.

The shell sets *addsuffix*, *argv*, *autologout*, *command*, *echo_style*, *edit*, *gid*, *group*, *home*, *loginsh*, *oid*, *path*, *prompt*, *prompt2*, *prompt3*, *shell*, *shlvl*, *tcsh*, *term*, *tty*, *uid*, *user* and *version* at startup; they do not change thereafter unless changed by the user. The shell updates *cwd*, *dirstack*, *owd* and *status* when necessary, and sets *logout* on logout.

The shell synchronizes *afsuser*, *group*, *home*, *path*, *shlvl*, *term* and *user* with the environment variables of the same names: whenever the environment variable changes the shell changes the corresponding shell variable to match (unless the shell variable is read-only) and vice versa. Note that although *cwd* and *PWD* have identical meanings, they are not synchronized in this manner, and that the shell automatically interconverts the different formats of *path* and *PATH*.

addsuffix (+)

If set, filename completion adds '/' to the end of directories and a space to the end of normal files when they are matched exactly. Set by default.

afsuser (+)
If set, *autologout*'s autolock feature uses its value instead of the local username for kerberos authentication.

ampm (+)
If set, all times are shown in 12-hour AM/PM format.

argv
The arguments to the shell. Positional parameters are taken from *argv*, i.e. '$1' is replaced by '$argv[1]', etc. Set by default, but usually empty in interactive shells.

autocorrect (+)
If set, the **spell-word** editor command is invoked automatically before each completion attempt.

autoexpand (+)
If set, the **expand-history** editor command is invoked automatically before each completion attempt.

autolist (+)
If set, possibilities are listed after an ambiguous completion. If set to 'ambiguous', possibilities are listed only when no new characters are added by completion.

autologout (+)
The first word is the number of minutes of inactivity before automatic logout. The optional second word is the number of minutes of inactivity before automatic locking. When the shell automatically logs out, it prints 'auto-logout', sets the variable logout to 'automatic' and exits. When the shell automatically locks, the user is required to enter his password to continue working. Five incorrect attempts result in automatic logout. Set to '60' (automatic logout after 60 minutes, and no locking) by default in login and superuser shells, but not if the shell thinks it is running under a window system (i.e. the *DISPLAY* environment variable is set), the tty is a pseudo-tty (pty) or the shell was not so compiled (see the *version* shell variable). See also the *afsuser* and *logout* shell variables.

backslash_quote (+)
If set, backslashes ('\') always quote '\', ''', and '"'. This may make complex quoting tasks easier, but it can cause syntax errors in *csh* scripts.

cdpath
A list of directories in which **cd** should search for subdirectories if they aren't found in the current directory.

command (+)
If set, the command which was passed to the shell with the *-c* flag (q.v.).

complete (+)
If set to 'enhance', completion 1) ignores case and 2) considers periods, hyphens and underscores ('.', '–' and '_') to be word separators and hyphens and underscores to be equivalent.

correct (+)
If set to 'cmd', commands are automatically spelling-corrected. If set to 'complete', commands are automatically completed. If set to 'all', the entire command line is corrected.

cwd
The full pathname of the current directory. See also the *dirstack* and *owd* shell variables.

dextract (+)
If set, 'pushd +**n**' extracts the **n**th directory from the directory stack rather than rotating it to the top.

dirsfile (+)
The default location in which 'dirs –S' and 'dirs –L' look for a history file. If unset, **~/.cshdirs** is used. Because only **~/.tcshrc** is normally sourced before **~/.cshdirs**, *dirsfile* should be set in **~/.tcshrc** rather than **~/.login**.

dirstack (+)
An array of all the directories on the directory stack. '$dirstack[1]' is the current working directory, '$dirstack[2]' the first directory on the stack, etc. Note that the current working directory is '$dirstack[1]' but '=0' in directory stack substitutions, etc. One can change the stack arbitrarily by setting *dirstack*, but the first element (the current working directory) is always correct. See also the *cwd* and *owd* shell variables.

dunique (+)
If set, **pushd** removes any instances of *name* from the stack before pushing it onto the stack.

echo
If set, each command with its arguments is echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively. Set by the **–x** command line option.

echo_style (+)
The style of the **echo** builtin. May be set to

bsd	Don't echo a newline if the first argument is '–n'.
sysv	Recognize backslashed escape sequences in echo strings.
both	Recognize both the '–n' flag and backslashed escape sequences; the default.
none	Recognize neither.

Set by default to 'both'. The BSD and System V options are described in the *echo* reference pages on the appropriate systems.

edit (+)
If set, the command-line editor is used. Set by default in interactive shells.

ellipsis (+)
If set, the '%c' / '%.' and '%C' prompt sequences (see the *prompt* shell

variable) indicate skipped directories with an ellipsis ('...') instead of '/<skipped>'.

fignore (+)
Lists file name suffixes to be ignored by completion.

filec In *tcsh*, completion is always used and this variable is ignored. If set in *csh*, filename completion is used.

gid (+) The user's real group ID.

group (+)
The user's group name.

histchars
A string value determining the characters used in *History substitution* (q.v.). The first character of its value is used as the history substitution character, replacing the default character '!'. The second character of its value replaces the character '^' in quick substitutions.

histdup (+)
Controls handling of duplicate entries in the history list. If set to 'all' only unique history events are entered in the history list. If set to 'prev' and the last history event is the same as the current command, then the current command is not entered in the history. If set to 'erase' and the same event is found in the history list, that old event gets erased and the current one gets inserted.

histfile (+)
The default location in which 'history –S' and 'history –L' look for a history file. If unset, **~/.history** is used. *histfile* is useful when sharing the same home directory between different machines, or when saving separate histories on different terminals. Because only **~/.tcshrc** is normally sourced before **~/.history**, *histfile* should be set in **~/.tcshrc** rather than **~/.login**.

histlit (+)
If set, builtin and editor commands and the *savehist* mechanism use the literal (unexpanded) form of lines in the history list. See also the **toggle-literal-history** editor command.

history The first word indicates the number of history events to save. The optional second word (+) indicates the format in which history is printed; if not given, '%h\t%T\t%R\n' is used. The format sequences are described below under *prompt*; note the variable meaning of '%R'. Set to '100' by default.

home Initialized to the home directory of the invoker. The filename expansion of '~' refers to this variable.

ignoreeof
If set to the empty string or '0' and the input device is a terminal, the **end-of-file** command (usually generated by the user by typing '^D' on an empty line) causes the shell to print 'Use "exit" to leave tcsh.' instead of exiting. This prevents the shell from accidentally being killed. If set to a number *n*, the shell ignores *n*-1 consecutive *end-of-file*s and exits on the *n*th. (+) If unset, '1' is used, i.e. the shell exits on a

single ‘^D’.

inputmode (+)
If set to ‘insert’ or ‘overwrite’, puts the editor into that input mode at the beginning of each line.

listflags (+)
If set to ‘x’, ‘a’ or ‘A’, or any combination thereof (e.g. ‘xA’), they are used as flags to **ls–F**, making it act like ‘ls –xF’, ‘ls –Fa’, ‘ls –FA’ or a combination (e.g. ‘ls –FxA’): ‘a’ shows all files (even if they start with a ‘.’), ‘A’ shows all files but ‘.’ and ‘..’, and ‘x’ sorts across instead of down. If the second word of *listflags* is set, it is used as the path to the *ls* command.

listjobs (+)
If set, all jobs are listed when a job is suspended. If set to ‘long’, the listing is in long format.

listlinks (+)
If set, the **ls–F** builtin command shows the type of file to which each symbolic link points.

listmax (+)
The maximum number of items which the **list-choices** editor command will list without asking first.

listmaxrows (+)
The maximum number of rows of items which the **list-choices** editor command will list without asking first.

loginsh (+)
Set by the shell if it is a login shell. Setting or unsetting it within a shell has no effect. See also *shlvl*.

logout (+)
Set by the shell to ‘normal’ before a normal logout, ‘automatic’ before an automatic logout, and ‘hangup’ if the shell was killed by a hangup signal (see *Signal handling*). See also the *autologout* shell variable.

mail The names of the files or directories to check for incoming mail, separated by whitespace, and optionally preceded by a numeric word. Before each prompt, if 10 minutes have passed since the last check, the shell checks each file and says ‘You have new mail.’ (or, if *mail* contains multiple files, ‘You have new mail in *name*.’) if the filesize is greater than zero in size and has a modification time greater than its access time.

If you are in a login shell, then no mail file is reported unless it has been modified after the time the shell has started up, in order to prevent redundant notifications. Most login programs will tell you whether or not you have mail when you log in.

If a file specified in *mail* is a directory, the shell will count each file within that directory as a separate message, and will report ‘You have **n** mails.’ or ‘You have *n* mails in *name*.’ as appropriate. This functionality is provided primarily for those systems which store mail in this manner, such as the Andrew Mail System.

If the first word of *mail* is numeric it is taken as a different mail checking interval, in seconds.

Under very rare circumstances, the shell may report 'You have mail.' instead of 'You have new mail.'

matchbeep (+)
If set to 'never', completion never beeps.
If set to 'nomatch', it beeps only when there is no match.
If set to 'ambiguous, it beeps when there are multiple matches.
If set to 'notunique', it beeps when there is one exact and other longer matches.
If unset, 'ambiguous' is used.

nobeep (+)
If set, beeping is completely disabled. See also *visiblebell*.

noclobber
If set, restrictions are placed on output redirection to insure that files are not accidentally destroyed and that '>>' redirections refer to existing files, as described in the "Input/output" section.

noglob If set, "Filename substitution" and "Directory stack substitution" (q.v.) are inhibited. This is most useful in shell scripts which do not deal with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nokanji (+)
If set and the shell supports Kanji (see the *version* shell variable), it is disabled so that the meta key can be used.

nonomatch
If set, a "Filename substitution" or "Directory stack substitution" (q.v.) which does not match any existing files is left untouched rather than causing an error. It is still an error for the substitution to be malformed, e.g. 'echo [' still gives an error.

nostat (+)
A list of directories (or glob-patterns which match directories; see "Filename substitution(cd) that should not be *stat*()ed during a completion operation. This is usually used to exclude directories which take too much time to *stat*(), for example **/afs**.

notify
If set, the shell announces job completions asynchronously. The default is to present job completions just before printing a prompt.

oid (+) The user's real organization ID. (Domain/OS only)

owd (+) The old working directory, equivalent to the '–' used by **cd** and **pushd**. See also the *cwd* and *dirstack* shell variables.

path A list of directories in which to look for executable commands. A null word specifies the current directory. If there is no *path* variable then only full path names will execute. *path* is set by the shell at startup from the *PATH* environment variable or, if *PATH* does not exist, to a system-dependent default something like '(/usr/local/bin /usr/bsd

/bin /usr/bin .)'. The shell may put '.' first or last in *path* or omit it entirely depending on how it was compiled; see the *version* shell variable. A shell which is given neither the **–c** nor the **–t** option hashes the contents of the directories in *path* after reading **~/.tcshrc** and each time *path* is reset. If one adds a new command to a directory in *path* while the shell is active, one may need to do a **rehash** for the shell to find it.

printexitvalue (+)
If set and an interactive program exits with a non-zero status, the shell prints 'Exit *status*'.

prompt The string which is printed before reading each command from the terminal. *prompt* may include any of the following formatting sequences (+), which are replaced by the given information:

%/ The current working directory.
%~ The current working directory, but with one's home directory represented by '~' and other users' home directories represented by '~user' as per *Filename substitution*. '~user' substitution happens only if the shell has already used '**~user**' in a pathname in the current session.
%c[[**0**]*n*], **%.**[[**0**]*n*]
The trailing component of the current working directory, or **n** trailing components if a digit *n* is given. If *n* begins with '0', the number of skipped components precede the trailing component(s) in the format '/<**skipped**>trailing'. If the *ellipsis* shell variable is set, skipped components are represented by an ellipsis so the whole becomes '...trailing'. '~' substitution is done as in '%~' above, but the '~' component is ignored when counting trailing components.
%C Like **%c**, but without '~' substitution.
%h, **%!**, **!**
The current history event number.
%M The full hostname.
%m The hostname up to the first '.'.
%S
Start (stop) standout mode.
%B (%b)
Start (stop) boldfacing mode.
%U (**%u**)
Start (stop) underline mode.
%t, **%@** The time of day in 12-hour AM/PM format.
%T Like **%t**, but in 24-hour format (but see the *ampm* shell variable).
%p The 'precise' time of day in 12-hour AM/PM format, with seconds.
%P Like **%p**, but in 24-hour format (but see the *ampm* shell variable).
c *c* is parsed as in **bindkey**.
^*c* *c* is parsed as in **bindkey**.
%% A single '%'.
%n The user name.

%d The weekday in 'Day' format.
%D The day in 'dd' format.
%w The month in 'Mon' format.
%W The month in 'mm' format.
%y The year in 'yy' format.
%Y The year in 'yyyy' format.
%l The shell's tty.
%L Clears from the end of the prompt to end of the display or the end of the line.
%# '>' (or the first character of the *promptchars* shell variable) for normal users, '#' (or the second character of *promptchars*) for the superuser.
%{*string***%}**
Includes *string* as a literal escape sequence. It should be used only to change terminal attributes and should not move the cursor location. This cannot be the last sequence in *prompt*.
%? The return code of the command executed just before the prompt.
%R In *prompt2*, the status of the parser. In *prompt3*, the corrected string. In *history*, the history string.

%B, **%S**, **%U** and **%{***string***%}** are available only in eight-bit-clean shells; see the *version* shell variable.

The bold, standout and underline sequences are often used to distinguish a superuser shell. For example,

```
> set prompt = "%m [%h] %B[%@]%b [%/] you rang? "
tut [37] [2:54pm] [/usr/accts/sys] you rang? _
```

Set by default to '%# ' in interactive shells.

prompt2 (+)
The string with which to prompt in **while** and **foreach** loops and after lines ending in '\'. The same format sequences may be used as in *prompt* (q.v.); note the variable meaning of '%R'. Set by default to '%R? ' in interactive shells.

prompt3 (+)
The string with which to prompt when confirming automatic spelling correction. The same format sequences may be used as in *prompt* (q.v.); note the variable meaning of '%R'. Set by default to 'CORRECT>%R (y | n | e)? ' in interactive shells.

promptchars (+)
If set (to a two-character string), the '%#' formatting sequence in the *prompt* shell variable is replaced with the first character for normal users and the second character for the superuser.

pushdtohome (+)
If set, **pushd** without arguments does 'pushd ~', like **cd**.

pushdsilent (+)
If set, **pushd** and **popd** do not print the directory stack.

recexact (+)
If set, completion completes on an exact match even if a longer match is possible.

recognize_only_executables (+)
If set, command listing displays only files in the path that are executable. Slow.

rmstar (+)
If set, the user is prompted before 'rm *' is executed.

savedirs (+)
If set, the shell does 'dirs –S' before exiting.

savehist
If set, the shell does 'history –S' before exiting. If the first word is set to a number, at most that many lines are saved. (The number must be less than or equal to *history*.) If the second word is set to 'merge', the history list is merged with the existing history file instead of replacing it (if there is one) and sorted by time stamp and the most recent events are retained. (+)

sched (+)
The format in which the **sched** builtin command prints scheduled events; if not given, '%h\t%T\t%R\n' is used. The format sequences are described above under *prompt*; note the variable meaning of '%R'.

shell The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Builtin and non-builtin command execution*.) Initialized to the (system-dependent) home of the shell.

shlvl (+)
The number of nested shells. Reset to 1 in login shells. See also *loginsh*.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status '1', all other builtin commands return status '0'.

symlinks (+)
Can be set to several different values to control symbolic link ('symlink') resolution. None of these have an effect under INTERIX, which does not yet support symbolic links.

- If set to **chase**, whenever the current directory changes to a directory containing a symbolic link, it is expanded to the real name of the directory to which the link points. This does not work for the user's home directory; this is a bug.
- If set to **ignore**, the shell tries to construct a current directory relative to the current directory before the link was crossed. This means that **cd**ing through a symbolic link and then 'cd ..'ing returns one to the original directory. This only affects builtin commands and filename completion.

- If set to **expand**, the shell tries to fix symbolic links by actually expanding arguments which look like path names. This affects any command, not just builtins. Unfortunately, this does not work for hard-to-recognize filenames, such as those embedded in command options. Expansion may be prevented by quoting. While this setting is usually the most convenient, it is sometimes misleading and sometimes confusing when it fails to recognize an argument which should be expanded. A compromise is to use 'ignore' and use the editor command **normalize-path** (bound by default to ^X-n) when necessary.

Some examples are in order. First, let's set up some play directories:

```
> cd /tmp
> mkdir from from/src to
> ln –s from/src to/dist
```

Here's the behavior with *symlinks* unset,

```
> cd /tmp/to/dist; echo $cwd
/tmp/to/dist
> cd ..; echo $cwd
/tmp/from
```

here's the behavior with *symlinks* set to 'chase',

```
> cd /tmp/to/dst; echo $cwd
/tmp/from/src
> cd ..; echo $cwd
/tmp/from
```

Here's the behavior with *symlinks* set to 'ignore',

```
> cd /tmp/to/dist; echo $cwd
/tmp/to/dst
> cd ..; echo $cwd
/tmp/to
```

and here's the behavior with *symlinks* set to 'expand'.

```
> cd /tmp/to/dist; echo $cwd
/tmp/to/dst
> cd ..; echo $cwd
/tmp/to
> cd /tmp/to/dist; echo $cwd
/tmp/to/dst
> cd ".."; echo $cwd
/tmp/from
> /bin/echo ..
/tmp/to
> /bin/echo ".."
..
```

Note that 'expand' expansion 1) works just like 'ignore' for builtins like *cd*, 2) is prevented by quoting, and 3) happens before filenames are passed to non-builtin commands.

tcsh (+)
The version number of the shell in the format 'R.VV.PP', where 'R' is the major release number, 'VV' the current version and 'PP' the patchlevel.

term
The terminal type. Usually set in **~/.login** as described under "Startup and shutdown".

time
If set to a number, then the **time** builtin (q.v.) executes automatically after each command which takes more than that many CPU seconds. If there is a second word, it is used as a format string for the output of the **time** builtin. (u) The following sequences may be used in the format string:

%U	The time the process spent in user mode in cpu seconds.
%S	The time the process spent in kernel mode in cpu seconds.
%E	The elapsed (wall clock) time in seconds.
%P	The CPU percentage computed as (%U + %S) / %E.
%W	Number of times the process was swapped.
%X	The average amount in (shared) text space used in Kbytes.
%D	The average amount in (unshared) data/stack space used in Kbytes.
%K	The total space used (%X + %D) in Kbytes.
%M	The maximum memory the process had in use at any time in Kbytes.
%F	The number of major page faults (page needed to be brought from disk).
%R	The number of minor page faults.
%I	The number of input operations.
%O	The number of output operations.
%r	The number of socket messages received.
%s	The number of socket messages sent.
%k	The number of signals received.
%w	The number of voluntary context switches (waits).
%c	The number of involuntary context switches.

Only the first four sequences are supported on systems without BSD resource limit functions. The default time format is '%Uu %Ss %E %P %X+%Dk %I+%Oio %Fpf+%Ww' for systems that support resource usage reporting and '%Uu %Ss %E %P' for systems that do not.

Note that the CPU percentage can be higher than 100% on multi-processors.

tperiod (+)
The period, in minutes, between executions of the **periodic** special alias.

tty (+) The name of the tty, or empty if not attached to one.

uid (+) The user's real user ID.

user The user's login name.

verbose

If set, causes the words of each command to be printed, after history substitution (if any). Set by the **–v** command line option.

version (+)

The version ID stamp. It contains the shell's version number (see *tcsh*), origin, release date, vendor, operating system and machine (see *VENDOR*, *OSTYPE* and *MACHTYPE*) and a comma-separated list of options which were set at compile time. Options which are set by default in the distribution are noted.

8b	The shell is eight bit clean; default
7b	The shell is not eight bit clean
nls	The system's NLS is used; default for systems with NLS
lf	Login shells execute **$OPENNT_ROOT/etc/csh.login** before instead of after **$OPENNT_ROOT/etc/csh.cshrc** and **~/.login** before instead of after **~/.tcshrc** and **~/.history**.
dl	'.' is put last in *path* for security; default
nd	'.' is omitted from *path* for security
vi	**vi**-style editing is the default rather than **emacs**
dtr	Login shells drop DTR when exiting
bye	**bye** is a synonym for **logout** and **log** is an alternate name for **watchlog**
al	*autologout* is enabled; default
kan	Kanji is used and the ISO character set is ignored, unless the *nokanji* shell variable is set
sm	The system's *malloc*( ) is used
hb	The '#!<program> <args>' convention is emulated when executing shell scripts
ng	The **newgrp** builtin is available
rh	The shell attempts to set the *REMOTEHOST* environment variable
afs	The shell verifies your password with the kerberos server if local authentication fails. The *afsuser* shell variable or the *AFSUSER* environment variable override your local username if set.

An administrator may enter additional strings to indicate differences in the local version.

visiblebell (+)

If set, a screen flash is used rather than the audible bell. See also *nobeep*.

watch (+)

A list of user / terminal pairs to watch for logins and logouts. If either the user is 'any' all terminals are watched for the given user and vice versa. Setting *watch* to '(any any)' watches all users and terminals. For example,

```
set watch = (george ttyd1 any console $user any)
```

reports activity of the user 'george' on ttyd1, any user on the console, and oneself (or a trespasser) on any terminal.

Logins and logouts are checked every 10 minutes by default, but the first word of *watch* can be set to a number to check every so many minutes. For example,

```
set watch = (1 any any)
```

reports any login/logout once every minute. For the impatient, the **log** builtin command triggers a *watch* report at any time. All current logins are reported (as with the **log** builtin) when *watch* is first set.

The *who* shell variable controls the format of *watch* reports.

who (+) The format string for *watch* messages. The following sequences are replaced by the given information:

%n	The name of the user who logged in/out.
%a	The observed action, i.e. 'logged on', 'logged off' or 'replaced *olduser* on'.
%l	The terminal (tty) on which the user logged in/out.
%M	The full hostname of the remote host, or 'local' if the login/logout was from the local host.
%m	The hostname of the remote host up to the first '.'. The full name is printed if it is an IP address or an X Window System display.

%M and %m are available only on systems which store the remote hostname in **/etc/utmp**. If unset, '%n has %a %l from %m.' is used, or '%n has %a %l.' on systems which don't store the remote hostname.

wordchars (+)
A list of non-alphanumeric characters to be considered part of a word by the **forward-word**, **backward-word** etc. editor commands. If unset, '*?_–.[]~=' is used.

ENVIRONMENT VARIABLES

Tcsh makes use of the following environment variables.

AFSUSER (+)
Equivalent to the *afsuser* shell variable.

COLUMNS
The number of columns in the terminal. See "Terminal management".

DISPLAY
Used by X Window System (see *X*). If set, the shell does not set *autologout* (q.v.).

EDITOR
The pathname to a default editor. See also the *VISUAL* environment variable and the **run-fg-editor** editor command.

GROUP (+)
Equivalent to the *group* shell variable.

HOME
Equivalent to the *home* shell variable.

HOST (+)
Initialized to the name of the machine on which the shell is running,

as determined by the *gethostname*( ) system call.

HOSTTYPE (+)
Initialized to the type of machine on which the shell is running, as determined at compile time. This variable is obsolete and will be removed in a future version.

HPATH (+)
A colon-separated list of directories in which the **run-help** editor command looks for command documentation.

LANG
Gives the preferred character environment. See "Native Language System support".

LC_CTYPE
If set, only ctype character handling is changed. See "Native Language System support".

LINES
The number of lines in the terminal. See "Terminal management".

MACHTYPE (+)
The machine type (microprocessor class or machine model), as determined at compile time.

NOREBIND (+)
If set, printable characters are not rebound to **self-insert-command**. See "Native Language System support".

OSTYPE (+)
The operating system, as determined at compile time.

PATH
A colon-separated list of directories in which to look for executables. Equivalent to the *path* shell variable, but in a different format.

PWD (+)
Equivalent to the *cwd* shell variable, but not synchronized to it; updated only after an actual directory change.

REMOTEHOST (+)
The host from which the user has logged in remotely, if this is the case and the shell is able to determine it. Set only if the shell was so compiled; see the *version* shell variable.

SHLVL (+)
Equivalent to the *shlvl* shell variable.

SYSTYPE (+)
The current system type. (Domain / OS only)

TERM
Equivalent to the *term* shell variable.

TERMCAP
The terminal capability string. See "Terminal management".

USER
Equivalent to the *user* shell variable.

VENDOR (+)
The vendor, as determined at compile time.

VISUAL
The pathname to a default full-screen editor. See also the *EDITOR* environment variable and the **run-fg-editor** editor command.

FILES

Tcsh makes use of the following files:

$OPENNT_ROOT/etc/csh.cshrc
Read first by every shell. INTERIX systems use **$OPENNT_ROOT/etc/cshrc**. ConvexOS, Stellix and Intel use **/etc/cshrc**. and NeXTs use **/etc/cshrc.std**. A/UX, AMIX, Cray and IRIX have no equivalent in *csh*, but read this file in *tcsh* anyway. Solaris 2.x does not have it either, but *tcsh* reads **$OPENNT_ROOT/etc/.cshrc**. (+)

$OPENNT_ROOT/etc/csh.login
Read by login shells after **$OPENNT_ROOT/etc/csh.cshrc**. INTERIX systems read **$OPENNT_ROOT/etc/csh.login**. ConvexOS, Stellix and Intel use **/etc/login**, NeXTs use **$OPENNT_ROOT/etc/login.std**, Solaris 2.x uses **/etc/.login** and A/UX, AMIX, Cray and IRIX use **/etc/cshrc**.

~/.tcshrc (+)
Read by every shell after **$OPENNT_ROOT/etc/csh.cshrc** or its equivalent.

~/.cshrc Read by every shell, if **~/.tcshrc** doesn't exist, after **$OPENNT_ROOT/etc/csh.cshrc** or its equivalent. This manual uses "**~/.tcshrc**" to mean "**~/.tcshrc** or, if **~/.tcshrc** is not found, **~/.cshrc.**"

~/.history
Read by login shells after **~/.tcshrc** if *savehist* is set, but see also *histfile*.

~/.login Read by login shells after **~/.tcshrc** or **~/.history**. The shell may be compiled to read **~/.login** before instead of after **~/.tcshrc** and **~/.history**; see the *version* shell variable.

~/.cshdirs (+)
Read by login shells after **~/.login** if *savedirs* is set, but see also *dirsfile*.

$OPENNT_ROOT/etc/csh.logout
Read by login shells at logout. INTERIX systems read **$OPENNT_ROOT/etc/csh.logout**. ConvexOS, Stellix and Intel use **/etc/logout** and NeXTs use **/etc/logout.std**. A/UX, AMIX, Cray and IRIX have no equivalent in *csh*, but read this file in *tcsh* anyway. Solaris 2.x does not have it either, but *tcsh* reads **/etc/.cshrc**. (+)

~/.logout
Read by login shells at logout after **$OPENNT_ROOT/etc/csh.logout** or its equivalent.

$OPENNT_ROOT/bin/sh
Used to interpret shell scripts not starting with a '#'.

$OPENNT_ROOT/tmp/sh*
Temporary file for **<<**.

$OPENNT_ROOT/etc/passwd

Source of home directories for '~name' substitutions.

The order in which startup files are read may differ if the shell was so compiled; see "Startup and shutdown" and the *version* shell variable.

NEW FEATURES (+)

This manual describes *tcsh* as a single entity, but experienced *csh* users will want to pay special attention to *tcsh*'s new features.

- A command-line editor, which supports GNU Emacs or *vi*-style key bindings. See "The command-line editor" and "Editor commands".
- Programmable, interactive word completion and listing. See "Completion and listing" and the **complete** and **uncomplete** builtin commands.
- "Spelling correction" (q.v.) of filenames, commands and variables.
- "Editor commands" (q.v.) which perform other useful functions in the middle of typed commands, including documentation lookup (**run-help**), quick editor restarting (**run-fg-editor**) and command resolution (**which-command**).
- An enhanced history mechanism. Events in the history list are time-stamped. See also the **history** command and its associated shell variables, the previously undocumented '#' event specifier and new modifiers under "History substitution," the ***-history**, **history-search-***, **i-search-***, **vi-search-*** and **toggle-literal-history** editor commands and the *histlit* shell variable.
- Enhanced directory parsing and directory stack handling. See the **cd**, **pushd**, **popd** and **dirs** commands and their associated shell variables, the description of "Directory stack substitution", the *dirstack*, *owd* and *symlinks* shell variables and the **normalize-command** and **normalize-path** editor commands.
- Negation in glob-patterns. See "Filename substitution".
- New "File inquiry operators" (q.v.) and a **filetest** builtin which uses them.
- A variety of "Automatic, periodic and timed events" (q.v.) including scheduled events, special aliases, automatic logout and terminal locking, command timing and watching for logins and logouts.
- Support for the Native Language System (see "Native Language System support"), OS variant features (see "OS variant support" and the *echo_style* shell variable) and system-dependent file locations (see "FILES").
- Extensive terminal-managment capabilities. See "Terminal management".
- New builtin commands including **builtins**, **hup**, **ls–F**, **newgrp**, **printenv**, **which** and **where** (q.v.).
- New variables that make useful information easily available to the shell. See the *gid*, *loginsh*, *oid*, *shlvl*, *tcsh*, *tty*, *uid* and *version* shell variables and the *HOST*, *REMOTEHOST*, *VENDOR*, *OSTYPE* and *MACHTYPE* environment variables.
- A new syntax for including useful information in the prompt string (see **prompt**). and special prompts for loops and spelling correction (see

prompt2 and **prompt3**).

- Read-only variables. See "Variable substitution".

BUGS

When a suspended command is restarted, the shell prints the directory it started in if this is different from the current directory. This can be misleading (i.e. wrong) as the job may have changed directories internally.

Shell builtin functions are not stoppable/restartable. Command sequences of the form 'a ; b ; c' are also not handled gracefully when stopping is attempted. If you suspend 'b', the shell will then immediately execute 'c'. This is especially noticeable if this expansion results from an **alias**. It suffices to place the sequence of commands in ()'s to force it to a subshell, i.e. '(a ; b ; c)'.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops are not placed in the history list. Control structures should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with '|', and to be used with '&' and ';' metasyntax.

foreach doesn't ignore here documents when looking for its **end**.

It should be possible to use the ':' modifiers on the output of command substitutions.

The screen update for lines longer than the screen width is very poor if the terminal cannot move the cursor up (i.e. terminal type 'dumb').

HPATH and *NOREBIND* don't need to be environment variables.

Glob-patterns which do not use '?', '*' or '[]' or which use '{}' or '~' are not negated correctly.

The single-command form of **if** does output redirection even if the expression is false and the command is not executed.

ls–F includes file identification characters when sorting filenames.

Report bugs to tcsh-bugs@mx.gw.com, preferably with fixes. If you want to help maintain and test tcsh, send mail to listserv@mx.gw.com with the text 'subscribe tcsh <your name>' on a line by itself in the body. You can also 'subscribe tcsh-bugs <your name>' to get all bug reports, or 'subscribe tcsh-diffs <your name>' to get the development list plus diffs for each patchlevel.

THE T IN TCSH

In 1964, DEC produced the PDP-6. The PDP-10 was a later re-implementation. It was re-christened the DECsystem-10 in 1970 or so when DEC brought out the second model, the KI10.

TENEX was created at Bolt, Beranek & Newman (a Cambridge, Mass. think tank) in 1972 as an experiment in demand-paged virtual memory operating

systems. They built a new pager for the DEC PDP-10 and created the OS to go with it. It was extremely successful in academia.

In 1975, DEC brought out a new model of the PDP-10, the KL10; they intended to have only a version of TENEX, which they had licensed from BBN, for the new box. They called their version TOPS-20 (their capitalization is trademarked). A lot of TOPS-10 users ('The OPerating System for PDP-10') objected; thus DEC found themselves supporting two incompatible systems on the same hardware--but then there were 6 on the PDP-11!

TENEX, and TOPS-20 to version 3, had command completion via a user-code-level subroutine library called ULTCMD. With version 3, DEC moved all that capability and more into the monitor ('kernel' for you modern types), accessed by the COMND% JSYS ('Jump to SYStem' instruction, the supervisor call mechanism [are my IBM roots also showing?]).

The creator of *tcsh* was impressed by this feature and several others of TENEX and TOPS-20, and created a version of *csh* which mimicked them.

LIMITATIONS

Words can be no longer than 1024 characters.

The system limits argument lists to 10240 characters.

The number of arguments to a command which involves filename expansion is limited to 1/6th the number of characters allowed in an argument list.

Command substitutions may substitute no more characters than are allowed in an argument list.

To detect looping, the shell restricts the number of **alias** substitutions on a single line to 20.

SEE ALSO

emacs, *ls*, *newgrp*, *sh*, *setpath*, *stty*, *su*, *tset*, *vi*, *X*

access(), *execve*(), *fork*(), *pipe*(), *stat*(), *umask*(), *vfork*(), *wait*()

malloc(), *setlocale*()

termcap

Introduction to the C Shell

VERSION

This manual documents tcsh 6.06.00 (Cornell) 1995-04-29.

AUTHORS

William Joy
: Original author of *csh*

J.E. Kulp, IIASA, Laxenburg, Austria
: Job control and directory stack features

Ken Greer, HP Labs, 1981
: File name completion

Mike Ellis, Fairchild, 1983
: Command name recognition/completion

Paul Placeway, Ohio State CIS Dept., 1983-1993
: Command line editor, prompt routines, new glob syntax and

numerous fixes and speedups

Karl Kleinpaste, CCI 1983-4
: Special aliases, directory stack extraction stuff, login/logout watch, scheduled events, and the idea of the new prompt format

Rayan Zachariassen, University of Toronto, 1984
: **ls–F** and **which** builtins and numerous bug fixes, modifications and speedups

Chris Kingsley, Caltech
: Fast storage allocator routines

Chris Grevstad, TRW, 1987
: Incorporated 4.3BSD *csh* into *tcsh*

Christos S. Zoulas, Cornell U. EE Dept., 1987-94
: Ports to HPUX, SVR2 and SVR3, a SysV version of getwd.c, SHORT_STRINGS support and a new version of sh.glob.c

James J Dempsey, BBN, and Paul Placeway, OSU, 1988
: A/UX port

Daniel Long, NNSC, 1988
: *wordchars*

Patrick Wolfe, Kuck and Associates, Inc., 1988
: **vi** mode cleanup

David C Lawrence, Rensselaer Polytechnic Institute, 1989
: *autolist* and ambiguous completion listing

Alec Wolman, DEC, 1989
: Newlines in the prompt

Matt Landau, BBN, 1989
: **~/.tcshrc**

Ray Moody, Purdue Physics, 1989
: Magic spacebar history expansion

Mordechai ????, Intel, 1989
: printprompt() fixes and additions

Kazuhiro Honda, Dept. of Computer Science, Keio University, 1989
: Automatic spelling correction and *prompt3*

Per Hedeland, Ellemtel, Sweden, 1990-
: Various bugfixes, improvements and manual updates

Hans J. Albertsson (Sun Sweden)
: *ampm*, **settc** and **telltc**

Michael Bloom
: Interrupt handling fixes

Michael Fine, Digital Equipment Corp
: Extended key support

Eric Schnoebelen, Convex, 1990
: Convex support, lots of *csh* bug fixes, save and restore of directory stack

Ron Flax, Apple, 1990
: A/UX 2.0 (re)port

Dan Oscarsson, LTH Sweden, 1990
: NLS support and simulated NLS support for non NLS sites, fixes

Johan Widen, SICS Sweden, 1990
: *shlvl*, Mach support, **correct-line**, 8-bit printing

Matt Day, Sanyo Icon, 1990
: POSIX termio support, SysV limit fixes

Jaap Vermeulen, Sequent, 1990-91
: Vi mode fixes, expand-line, window change fixes, Symmetry port

Martin Boyer, Institut de recherche d'Hydro-Quebec, 1991
: *autolist* beeping options, modified the history search to search for the whole string from the beginning of the line to the cursor.

Scott Krotz, Motorola, 1991
: Minix port

David Dawes, Sydney U. Australia, Physics Dept., 1991
: SVR4 job control fixes

Jose Sousa, Interactive Systems Corp., 1991
: Extended *vi* fixes and *vi* delete command

Marc Horowitz, MIT, 1991
: ANSIfication fixes, new exec hashing code, imake fixes, **where**

Bruce Sterling Woodcock, sterling@netcom.com, 1991-1995
: ETA and Pyramid port, Makefile and lint fixes, **ignoreeof**=n addition, and various other portability changes and bug fixes

Jeff Fink, 1992
: **complete-word-fwd** and **complete-word-back**

Harry C. Pulley, 1992
: Coherent port

Andy Phillips, Mullard Space Science Lab U.K., 1992
: VMS-POSIX port

Beto Appleton, IBM Corp., 1992
: Walking process group fixes, *csh* bug fixes, POSIX file tests, POSIX SIGHUP

Scott Bolte, Cray Computer Corp., 1992
: CSOS port

Kaveh R. Ghazi, Rutgers University, 1992
: Tek, m88k, Titan and Masscomp ports and fixes

Mark Linderman, Cornell University, 1992
: OS/2 port

Mika Liljeberg, liljeber@kruuna.Helsinki.FI, 1992
: Linux port

Tim P. Starrin, NASA Langley Research Center Operations, 1993
: Read-only variables

Dave Schweisguth, Yale University, 1993-4
: New manpage and tcsh.man2html

Larry Schwimmer, Stanford University, 1993
: AFS and HESIOD patches

Luke Mewburn, RMIT University, 1994
: Enhanced directory printing in prompt

THANKS TO

Bryan Dunlap, Clayton Elwell, Karl Kleinpaste, Bob Manson, Steve Romig, Diana Smetters, Bob Sutterfield, Mark Verber, Elizabeth Zwicky and all the other people at Ohio State for suggestions and encouragement

All the people on the net, for putting up with, reporting bugs in, and suggesting new additions to each and every version

Richard M. Alderson III, for writing the 'T in tcsh' section

NAME

tee – pipe fitting (duplicate standard input)

SYNOPSIS

`tee [-ai] [file ...]`

DESCRIPTION

The *tee* utility copies standard input to standard output, making a copy in zero or more files. The output is unbuffered.

The following options are available:

–a Append the output to the files rather than overwriting them.

–i Ignore the `SIGINT` signal.

The *file* is a pathname of an output file.

The *tee* utility takes the default action for all signals, except in the event of the **–i** option.

DIAGNOSTICS

The *tee* utility exits 0 on success, and >0 if an error occurs.

NAME

telnet – user interface to the TELNET protocol

SYNOPSIS

```
telnet [-8acdELr] [-e escapechar] [-l user]
       [-n tracefile] [-S tos] [-X authtype]
       [host [port]]
```

DESCRIPTION

The *telnet* command is used to communicate with another host using the TELNET protocol. If *telnet* is invoked without the *host* argument, it enters command mode, indicated by its prompt (`telnet>`.) In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command with those arguments.

Options:

–8 Specifies an 8-bit data path. This causes an attempt to negotiate the TELNET BINARY option on both input and output.

–a Attempt automatic login. Currently, this sends the user name via the *USER* variable of the ENVIRON option if supported by the remote system. The name used is that of the current user as returned by *getlogin*( ) if it agrees with the current user ID, otherwise it is the name associated with the user ID.

–c Disables the reading of the user's **.telnetrc** file. (See the **toggle skiprc** command on this man page.)

–d Sets the initial value of the **debug** toggle to TRUE.

–E Stops any character from being recognized as an escape character.

–e*escape_char*
Sets the initial *telnet* escape character to *escape_char*. If *escape_char* is omitted, then there will be no escape character.

–L Specifies an 8-bit data path on output. This causes the BINARY option to be negotiated on output.

–l*user* When connecting to the remote system, if the remote system understands the ENVIRON option, then *user* will be sent to the remote system as the value for the variable *USER*. This option implies the **–a** option. This option may also be used with the **open** command.

–n*tracefile*
Opens *tracefile* for recording trace information. See the **set tracefile** command below.

–r Specifies a user interface similar to *rlogin*. In this mode, the escape character is set to the tilde (~) character, unless modified by the **–e** option.

–S*tos* Sets the IP type-of-service (TOS) option for the telnet connection to the value *tos*, which can be a numeric TOS value or, on systems that support it, a symbolic TOS name found in the **/etc/iptos** file.

–X*atype* Disables the *atype* type of authentication.

host Indicates the official name, an alias, or the Internet address of a remote host.

port Indicates a port number (address of an application). If a number is not specified, the default *telnet* port is used.

When in rlogin mode, a line of the form ~. disconnects from the remote host; ~ is the telnet escape character. Similarly, the line ~^Z suspends the telnet session. The line ~^] escapes to the normal telnet escape prompt.

Once a connection has been opened, *telnet* will attempt to enable the TELNET LINEMODE option. If this fails, then *telnet* will revert to one of two input modes: either "character at a time" or "old line by line" depending on what the remote system supports.

- When LINEMODE is enabled, character processing is done on the local system, under the control of the remote system. When input editing or character echoing is to be disabled, the remote system will relay that information. The remote system will also relay changes to any special characters that happen on the remote system, so that they can take effect on the local system.
- In "character at a time" mode, most text typed is immediately sent to the remote host for processing.
- In "old line by line" mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The "local echo character" (initially "^E") may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

If the LINEMODE option is enabled, or if the **localchars** toggle is TRUE (the default for "old line by line"; see below), the user's *quit*, *intr*, and *flush* characters are trapped locally, and sent as TELNET protocol sequences to the remote side. If LINEMODE has ever been enabled, then the user's *susp* and *eof* are also sent as TELNET protocol sequences, and *quit* is sent as a TELNET ABORT instead of BREAK. There are options (see **toggle autoflush** and **toggle autosynch** below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of *quit* and *intr*).

While connected to a remote host, *telnet* command mode may be entered by typing the *telnet* "escape character" (initially "^]"). When in command mode, the normal terminal editing conventions are available.

The following *telnet* commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, **unset**, **slc**, **environ**, and **display** commands).

auth *argument ...*

The **auth** command manipulates the information sent through the TELNET AUTHENTICATE option. Valid arguments for the **auth** command are as follows:

disable *type*

Disables the specified type of authentication. To obtain a list

of available types, use the **auth disable ?** command.

enable *type*
Enables the specified type of authentication. To obtain a list of available types, use the **auth enable ?** command.

status
Lists the current status of the various types of authentication.

close
Close a TELNET session and return to command mode.

display *argument ...*
Displays all, or some, of the **set** and **toggle** values (see below).

encrypt
The encrypt command manipulates the information sent through the TELNET ENCRYPT option.

Note: Because of export controls, the TELNET ENCRYPT option is not supported outside of the United States and Canada.

Valid arguments for the encrypt command are as follows:

disable Disables the specified type of encryption. If you omit the input and output, both input and output are disabled. To obtain a list of available types, use the **encrypt disable ?** command.
enable Enables the specified type of encryption. If you omit input and output, both input and output are enabled. To obtain a list of available types, use the **encrypt enable ?** command.
input This is the same as the **encrypt start input** command.
-input This is the same as the **encrypt stop input** command.
output This is the same as the **encrypt start output** command.
-output This is the same as the **encrypt stop output** command.
start [**input** | **output**]
Attempts to start encryption. If you omit **input** and **output**, both input and output are enabled. To obtain a list of available types, use the **encrypt enable ?** command.
status **Lists the current status of encryption.**
stop [input | output]
Stops encryption. If you omit input and output, encryption is on both input and output.
type *type*
Sets the default type of encryption to be used with later **encrypt start** or **encrypt stop** commands.

environ *arguments...*
The **environ** command is used to manipulate the the variables that may be sent through the TELNET ENVIRON option. The initial set of variables is taken from the user's environment, with only the *DISPLAY* and *PRINTER* variables being exported by default. The *USER* variable is also exported if the **–a** or **–l** options are used.

Valid arguments for the **environ** command are:

define Define the variable *variable* to have a value of *value.* Any

variables defined by this command are automatically exported. The *value* may be enclosed in single or double quotes so that tabs and spaces may be included.

undefine
Remove *variable* from the list of environment variables.

export Mark the variable *variable* to be exported to the remote side.

unexport
Mark the variable *variable* to not be exported unless explicitly asked for by the remote side.

list List the current set of environment variables. Those marked with a * will be sent automatically, other variables will only be sent if explicitly requested.

? Prints out help information for the **environ** command.

logout

Sends the TELNET LOGOUT option to the remote side. This command is similar to a **close** command; however, if the remote side does not support the LOGOUT option, nothing happens. If, however, the remote side does support the LOGOUT option, this command should cause the remote side to close the TELNET connection. If the remote side also supports the concept of suspending a user's session for later reattachment, the logout argument indicates that you should terminate the session immediately.

mode *type*

Type is one of several options, depending on the state of the TELNET session. The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

character
Disable the TELNET LINEMODE option, or, if the remote side does not understand the LINEMODE option, then enter "character at a time" mode.

line Enable the TELNET LINEMODE option, or, if the remote side does not understand the LINEMODE option, then attempt to enter "old-line-by-line" mode.

isig
–isig Attempt to enable (**–isig** disables) the TRAPSIG mode of the LINEMODE option. This requires that the LINEMODE option be enabled.

edit
–edit Attempt to enable (**–edit** disables) the EDIT mode of the LINEMODE option. This requires that the LINEMODE option be enabled.

softtabs
–softtabs
Attempt to enable (**–softtabs** disables) the SOFT_TAB mode of the LINEMODE option. This requires that the LINEMODE option be enabled.

litecho
–litecho
Attempt to enable (**–litecho** disables) the LIT_ECHO mode

of the LINEMODE option. This requires that the LINEMODE option be enabled.

? Prints out help information for the **mode** command.

open *host* [[**–l**] *user*] [**-***port*]

Open a connection to the named host. If no port number is specified, *telnet* will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts*) or an Internet address specified in the "dot notation" (see *inet*( )). The **–l** option may be used to specify the user name to be passed to the remote system via the ENVIRON option. When connecting to a non-standard port, *telnet* omits any automatic initiation of TELNET options. When the port number is preceded by a minus sign, the initial option negotiation is done. After establishing a connection, the file **.telnetrc** in the user's home directory is opened. Lines beginning with a # are comment lines. Blank lines are ignored. Lines that begin without white space are the start of a machine entry. The first thing on the line is the name of the machine that is being connected to. The rest of the line, and successive lines that begin with white space are assumed to be *telnet* commands and are processed as if they had been typed in manually to the *telnet* command prompt.

quit Close any open TELNET session and exit *telnet*. An end of file (in command mode) will also close a session and exit.

send Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

abort Sends the TELNET ABORT (Abort processes) sequence.

ao Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output *from* the remote system *to* the user's terminal.

ayt Sends the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.

brk Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.

ec Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

eof Sends the TELNET EOF (End Of File) sequence.

eor Sends the TELNET EOR (End of Record) sequence.

escape Sends the current *telnet* escape character (initially "^").

ga Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.

getstatus

If the remote side supports the TELNET STATUS command, **getstatus** will send the subnegotiation to request that the server send its current option status.

ip Sends the TELNET IP (Interrupt Process) sequence, which

should cause the remote system to abort the currently running process.

nop Sends the TELNET NOP (No OPeration) sequence.

susp Sends the TELNET SUSP (SUSPend process) sequence.

synch Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a BSD 4.2 system -- if it doesn't work, a lower case "r" may be echoed on the terminal).

do
dont
will
wont Sends the TELNET DO *cmd* sequence. *Cmd* can be either a decimal number between 0 and 255, or a symbolic name for a specific TELNET command. *Cmd* can also be either **help** or **?** to print out help information, including a list of known symbolic names.

? Prints out help information for the **send** command.

set

unset

The **set** command will set any one of a number of *telnet* variables to a specific value or to TRUE. The special value **off** turns off the function associated with the variable, this is equivalent to using the **unset** command. The **unset** command will disable or set to FALSE any of the specified functions. The values of variables may be interrogated with the **display** command. The variables which may be set or unset, but not toggled, are listed here. In addition, any of the variables for the **toggle** command may be explicitly set or unset using the **set** and **unset** commands.

ayt If TELNET is in **localchars** mode, or LINEMODE is enabled, and the status character is typed, a TELNET AYT sequence (see **send ayt** preceding) is sent to the remote host. The initial value for the "Are You There" character is the terminal's status character.

echo In "line by line" mode, this value toggles between doing local "line by line" mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password). The initial value is "^E".

eof If *telnet* is operating in LINEMODE or "old line by line" mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's **eof** character.

erase If *telnet* is in **localchars** mode (see **toggle localchars** below), *and* if *telnet* is operating in "character at a time" mode, then when this character is typed, a TELNET EC sequence (see **send ec** above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase**

character.

escape This is the *telnet* escape character which causes entry into *telnet* command mode (when connected to a remote system). The initial value is "^[".

flushoutput
If *telnet* is in **localchars** mode (see **toggle localchars** below) and the **flushoutput** character is typed, a TELNET AO sequence (see **send ao** above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

forw1
forw2 If TELNET is operating in LINEMODE, these are the characters that, when typed, cause partial lines to be forwarded to the remote system. The initial value for the forwarding characters are taken from the terminal's eol and eol2 characters.

interrupt
If *telnet* is in **localchars** mode (see **toggle localchars** below) and the **interrupt** character is typed, a TELNET IP sequence (see **send ip** above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

kill If *telnet* is in **localchars** mode (see **toggle localchars** below), **and** if *telnet* is operating in "character at a time" mode, then when this character is typed, a TELNET EL sequence (see **send el** above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

lnext If *telnet* is operating in LINEMODE or "old line by line" mode, then this character is taken to be the terminal's **lnext** character. The initial value for the lnext character is taken to be the terminal's **lnext** character.

quit If *telnet* is in **localchars** mode (see **toggle localchars** below) and the **quit** character is typed, a TELNET BRK sequence (see **send brk** above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

reprint If *telnet* is operating in LINEMODE or "old line by line" mode, then this character is taken to be the terminal's **reprint** character. The initial value for the reprint character is taken to be the terminal's **reprint** character.

rlogin This is the rlogin escape character. If set, the normal TELNET escape character is ignored unless it is preceded by this character at the beginning of a line. This character, at the beginning of a line followed by a "." closes the connection; when followed by a ^Z it suspends the telnet command. The initial state is to disable the rlogin escape character.

start If the TELNET TOGGLE-FLOW-CONTROL option has been enabled, then this character is taken to be the terminal's **start** character. The initial value for the kill character is taken to be the terminal's **start** character.

stop If the TELNET TOGGLE-FLOW-CONTROL option has been enabled, then this character is taken to be the terminal's **stop** character. The initial value for the kill character is taken to be the terminal's **stop** character.

susp If *telnet* is in **localchars** mode, or LINEMODE is enabled, and the **suspend** character is typed, a TELNET SUSP sequence (see **send susp** above) is sent to the remote host. The initial value for the suspend character is taken to be the terminal's **suspend** character.

tracefile
This is the file to which the output, caused by **netdata** or **option** tracing being TRUE, will be written. If it is set to '–' then tracing information will be written to standard output (the default).

worderase
If *telnet* is operating in LINEMODE or "old line by line" mode, then this character is taken to be the terminal's **worderase** character. The initial value for the worderase character is taken to be the terminal's **worderase** character.

? Displays the legal **set** (**unset**) commands.

slc *state* The **slc** command (Set Local Characters) is used to set or change the state of the the special characters when the TELNET LINEMODE option has been enabled. Special characters are characters that get mapped to TELNET commands sequences (like **ip** or **quit)** or line editing characters (like **erase** and **kill).** By default, the local special characters are exported.

check Verify the current settings for the current special characters. The remote side is requested to send all the current special character settings, and if there are any discrepancies with the local side, the local side will switch to the remote value.

export Switch to the local defaults for the special characters. The local default characters are those of the local terminal at the time when *telnet* was started.

import Switch to the remote defaults for the special characters. The remote default characters are those of the remote system at the time when the TELNET connection was established.

? Prints out help information for the **slc** command.

status
Show the current status of *telnet*. This includes the peer one is connected to, as well as the current mode.

toggle *toggle*
Toggle (between TRUE and FALSE) various flags that control how *telnet* responds to events. These flags may be set explicitly to TRUE or FALSE using the **set** and **unset** commands listed above. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

authdebug
Turn on debugging information for the authentication code.

autoflush

If **autoflush** and **localchars** are both TRUE, then when the **ao**, or **quit** characters are recognized (and transformed into TELNET sequences; see **set** above for details), *telnet* refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET TIMING MARK option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an *stty noflsh*, otherwise FALSE (see *stty*).

autodecrypt

When the TELNET ENCRYPT option is negotiated, by default the actual encryption (decryption) of the data stream does not start automatically. The autoencrypt (autodecrypt) command states that encryption of the output (input) stream should be enabled as soon as possible.

Note: Because of export controls, the TELNET ENCRYPT option is not supported outside the United States and Canada.

autologin

If the remote side supports the TELNET AUTHENTICATION option TELNET attempts to use it to perform automatic authentication. If the AUTHENTICATION option is not supported, the user's login name are propagated through the TELNET ENVIRON option. This command is the same as specifying *a* option on the **open** command.

autosynch

If **autosynch** and **localchars** are both TRUE, then when either the **intr** or **quit** characters is typed (see **set** above for descriptions of the **intr** and **quit** characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

binary Enable or disable the TELNET BINARY option on both input and output.

inbinary

Enable or disable the TELNET BINARY option on input.

outbinary

Enable or disable the TELNET BINARY option on output.

crlf If this is TRUE, then carriage returns will be sent as <CR><LF>. If this is FALSE, then carriage returns will be send as <CR><NUL>. The initial value for this toggle is FALSE.

crmod Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this

toggle is FALSE.

debug Toggle socket level debugging (useful only to the super-user). The initial value for this toggle is FALSE.

encdebug
Turn on debugging information for the encryption code.

localchars
If this is TRUE, then the **flush**, **interrupt**, **quit**, **erase**, and **kill** characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively **ao**, **ip**, **brk**, **ec**, and **el**; see **send** above). The initial value for this toggle is TRUE in "old line by line" mode, and FALSE in "character at a time" mode. When the lINEMODE option is enabled, the value of **localchars** is ignored, and assumed to always be TRUE. If lINEMODE has ever been enabled, then **quit** is sent as **abort**, and **eof** and **suspend** are sent as **eof** and **susp,** see **send** above).

netdata Toggle the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

options Toggle the display of some internal *telnet* protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

prettydump
When the **netdata** toggle is enabled, if **prettydump** is enabled the output from the **netdata** command will be formatted in a more user readable format. Spaces are put between each character in the output, and the beginning of any TELNET escape sequence is preceded by a '*' to aid in locating them.

skiprc When the skiprc toggle is TRUE, TELNET skips the reading of the **.telnetrc** file in the user's home directory when connections are opened. The initial value for this toggle is FALSE.

termdata
Toggle the display of all terminal data (in hexadecimal format). The initial value for this toggle is FALSE.

verbose_encrypt
When the **verbose_encrypt** toggle is TRUE, TELNET prints out a message each time encryption is enabled or disabled. The initial value for this toggle is FALSE. Note: Because of export controls, data encryption is not supported outside of the United States and Canada.

? Displays the legal **toggle** commands.

z Suspend *telnet*. This command only works when the user is using the *csh*.

! [*command*]
Execute a single command in a subshell on the local system. If *command* is omitted, then an interactive subshell is invoked.

? [*command*]
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information for just

that command.

ENVIRONMENT VARIABLES

Telnet uses at least the *HOME, SHELL, DISPLAY,* and *TERM* environment variables. Other environment variables may be propagated to the other side via the TELNET ENVIRON option.

FILES

~/.telnetrc

user customized telnet startup values

NOTES

On some remote systems, echo has to be turned off manually when in "old line by line" mode.

In "old line by line" mode or LINEMODE the terminal's eof character is only recognized (and sent to the remote system) when it is the first character on a line.

SEE ALSO

telnetd, service

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

```
telnetd [-hilKknQqU] [-a authmode]
        [-D debugmode] [-debug [port]]
        [-X authtype]
```

DESCRIPTION

The *telnetd* command is a server which supports the DARPA standard TELNET virtual terminal protocol. *Telnetd* is normally invoked by the internet server *inetd* for requests to connect to the TELNET port (see the **–i** option). On an INTERIX system, it can also be run as a service (see the *service* utility). The **–debug** option may be used to start up *telnetd* manually. If started up this way, *port* may be specified to run *telnetd* on an alternate TCP port number.

The *telnetd* command accepts the following options:

–a*authmode*

This option may be used for specifying what mode should be used for authentication. Note that this option is only useful if *telnetd* has been compiled with support for the AUTHENTICATION option. There are several valid values for *authmode:*

debug Turns on authentication debugging code.

user Only allow connections when the remote user can provide valid authentication information to identify the remote user, and is allowed access to the specified account without providing a password.

valid Only allow connections when the remote user can provide valid authentication information to identify the remote user. The *login* command will provide any additional user verification needed if the remote user is not allowed automatic access to the specified account.

other Only allow connections that supply some authentication information. This option is currently not supported by any of the existing authentication mechanisms, and is thus the same as specifying **–a valid**.

none This is the default state. Authentication information is not required. If no or insufficient authentication information is provided, then the *login* program will provide the necessary user verification.

off This disables the authentication code. All user verification will happen through the *login* program.

–D *debugmode*

This option may be used for debugging purposes. This allows *telnetd* to print out debugging information to the connection, allowing the user to see what *telnetd* is doing. There are several possible values for *debugmode:*

options Prints information about the negotiation of TELNET options.

report Prints the *options* information, plus some additional

information about what processing is going on.

netdata Displays the data stream received by *telnetd.*

ptydata Displays data written to the pty.

exercise

Has not been implemented yet.

–debug Enables debugging on each socket created by *telnetd* (see SO_DEBUG in *socket*( )).

–h Disables the printing of host-specific information before login has been completed.

–i Causes *telnetd* to assume that stdin, stdout, and stderr have been inherited from the parent. This allows *telnetd* to be started from *inetd.*

–K Turns on kludge linemode (see also **–k**). If the client responds with WILL TIMING-MARK in response to a DO TIMING-MARK), *telnetd* assumes that the client supports kludge linemode. Unfortunately, many clients which do not support kludge linemode pass the heuristic. The **–K** option is most useful if you are certain that all clients will support kludge linemode.

–k If the **–k** option is specified, then if the remote client does not support the LINEMODE option, then *telnetd* will operate in character-at-a-time mode. (The server will only go into kludge linemode if the the remote client requests it, by sending DONT SUPPRESS-GO-AHEAD and DONT ECHO.) The **–k** option is most useful when there are remote clients that do not support kludge linemode, but pass the heuristic (if they respond with WILL TIMING-MARK in response to a DO TIMING-MARK) for kludge linemode support. Most clients pass the heuristic even if they don't support kludge linemode, so the **–k** behavior is the default behavior for *telnetd.* Without this option, certain telnet clients (such as those on Solaris systems before 2.4) behave poorly.

–l Specifies line mode. Tries to force clients to use line-at-a-time mode. If the LINEMODE option is not supported, it will go into kludge linemode.

–n Disable TCP keep-alives. Normally *telnetd* enables the TCP keep-alive mechanism to probe connections that have been idle for some period of time to determine if the client is still there, so that idle connections from machines that have crashed or can no longer be reached may be cleaned up.

–Q Sets really quiet mode. Really quiet mode suppresses the entire login banner, unlike **–q**, which only suppresses part of the banner.

–q Sets quiet mode. Quiet mode suppresses the banner describing default login domains; *telnetd* still prints the portion of the banner with the operating system, version, machine name and terminal.

–U This option causes *telnetd* to refuse connections from addresses that cannot be mapped back into a symbolic name via the *gethostbyaddr*( ) routine.

–X*authtype*

This option is only valid if *telnetd* has been built with support for the authentication option; it is not valid for the INTERIX *telnetd*.) It disables the use of *authtype* authentication, and can be used to temporarily disable a specific authentication type without having to recompile *telnetd.*

Telnetd operates by allocating a pseudo-terminal device (see *pty*(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as stdin, stdout and stderr. *Telnetd* manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the remote client and the login process.

When a TELNET session is started up, *telnetd* sends TELNET options to the client side indicating a willingness to do the following TELNET options, which are described in more detail below:

```
DO AUTHENTICATION   WILL SUPPRESS GO AHEAD
WILL ENCRYPT        DO ECHO
DO TERMINAL TYPE    DO LINEMODE
DO TSPEED           DO NAWS
DO XDISPLOC         WILL STATUS
DO NEW-ENVIRON      DO LFLOW
DO ENVIRON          DO TIMING-MARK
```

The pseudo-terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS and CRMOD enabled (see *tty*).

Telnet Options

Telnetd has support for enabling locally the following TELNET options:

WILL ECHO

When the LINEMODE option is enabled, a WILL ECHO or WONT ECHO will be sent to the client to indicate the current state of terminal echoing. When terminal echo is not desired, a WILL ECHO is sent to indicate that *telnetd* will take care of echoing any data that needs to be echoed to the terminal, and then nothing is echoed. When terminal echo is desired, a WONT ECHO is sent to indicate that *telnetd* will not be doing any terminal echoing, so the client should do any terminal echoing that is needed.

WILL BINARY

Indicates that the client is willing to send a 8 bits of data, rather than the normal 7 bits of the Network Virtual Terminal.

WILL SGA

Indicates that it will not be sending IAC GA, go ahead, commands.

WILL STATUS

Indicates a willingness to send the client, upon request, of the current status of all TELNET options.

WILL TIMING-MARK

Whenever a DO TIMING-MARK command is received, it is always responded to with a WILL TIMING-MARK

WILL LOGOUT

When a DO LOGOUT is received, a WILL LOGOUT is sent in response, and the TELNET session is shut down.

WILL ENCRYPT
: Only sent if *telnetd* is compiled with support for data encryption, and indicates a willingness to decrypt the data stream.

Telnetd has support for enabling remotely the following TELNET options:

DO BINARY
: Sent to indicate that *telnetd* is willing to receive an 8 bit data stream.

DO LFLOW
: Requests that the client handle flow control characters remotely.

DO ECHO
: This is not really supported, but is sent to identify a 4.2BSD *telnet* client, which will improperly respond with WILL ECHO. If a WILL ECHO is received, a DONT ECHO will be sent in response.

DO TERMINAL-TYPE
: Indicates a desire to be able to request the name of the type of terminal that is attached to the client side of the connection.

DO SGA
: Indicates that it does not need to receive IAC GA, the go ahead command.

DO NAWS
: Requests that the client inform the server when the window (display) size changes.

DO TERMINAL-SPEED
: Indicates a desire to be able to request information about the speed of the serial line to which the client is attached.

DO XDISPLOC
: Indicates a desire to be able to request the name of the X windows display that is associated with the telnet client.

DO NEW-ENVIRON
: Indicates a desire to be able to request environment variable information, as described in RFC 1572.

DO ENVIRON
: Indicates a desire to be able to request environment variable information, as described in RFC 1408.

DO LINEMODE
: Only sent if *telnetd* is compiled with support for linemode, and requests that the client do line-by-line processing. This is enabled in INTERIX.

DO TIMING-MARK
: Only sent if *telnetd* is compiled with support for both linemode and kludge linemode, and the client responded with WONT LINEMODE. If the client responds with WILL TM, the it is assumed that the client supports kludge linemode. Note that the **–k** option can be used to disable this.

DO AUTHENTICATION
: Only sent if *telnetd* is compiled with support for authentication, and indicates a willingness to receive authentication information for automatic login.

DO ENCRYPT

Only sent if *telnetd* is compiled with support for data encryption, and indicates a willingness to decrypt the data stream.

DIAGNOSTICS

The *telnetd* program may exit with one of these values:

0 *telnetd* exited normally.

1 *telnetd* exited due to a usage or socket error.

2 *telnetd* exited due to a license problem.

For exit statuses 1 and 2, additional messages may be printed to standard error.

NOTES

Do not mount drives using *NET.EXE* from a session controlled by this daemon. See the *service* reference page for more discussion.

The default domain presented in the login banner can be changed by setting the Windows NT registry variable **HKEY_LOCAL_MACHINE** | **SOFTWARE** | **Softway Systems** | **OpenNT** | **LoginDefaultDomain** to the name of the domain.

If the *telnetd* is running on a Windows NT workgroup server without a domain, set the value of **LoginDefaultDomain** to an empty string. If you do not do this, users will have to type "*+loginname*" to login to the server.

STANDARDS

RFC-854	TELNET PROTOCOL SPECIFICATION
RFC-855	TELNET OPTION SPECIFICATIONS
RFC-856	TELNET BINARY TRANSMISSION
RFC-857	TELNET ECHO OPTION
RFC-858	TELNET SUPPRESS GO AHEAD OPTION
RFC-859	TELNET STATUS OPTION
RFC-860	TELNET TIMING MARK OPTION
RFC-861	TELNET EXTENDED OPTIONS - lIST OPTION
RFC-885	TELNET END OF RECORD OPTION
RFC-1073	Telnet Window Size Option
RFC-1079	Telnet Terminal Speed Option
RFC-1091	Telnet Terminal-Type Option
RFC-1096	Telnet X Display Location Option
RFC-1123	Requirements for Internet Hosts -- Application and Support
RFC-1184	Telnet Linemode Option
RFC-1372	Telnet Remote Flow Control Option
RFC-1416	Telnet Authentication Option
RFC-1411	Telnet Authentication: Kerberos Version 4
RFC-1412	Telnet Authentication: SPX
RFC-1571	Telnet Environment Option Interoperability Issues
RFC-1572	Telnet Environment Option

BUGS

Some TELNET commands are only partially implemented.

Because of bugs in the original 4.2 BSD *telnet*, *telnetd* performs some dubious protocol exchanges to try to discover if the remote client is, in fact, a 4.2 BSD *telnet*.

Binary mode has no common interpretation except between similar operating systems.

The terminal type name received from the remote client is converted to lower case.

Telnetd never sends TELNET IAC GA (go ahead) commands.

SEE ALSO

login, *rlogin*, *rlogind*, *service*, *telnet*

NAME

test – condition evaluation utility

SYNOPSIS

`test` *`expression`*

DESCRIPTION

The *test* utility evaluates the expression and, if it evaluates to true, returns a zero (true) exit status; otherwise it returns 1 (false). If there is no expression, test also returns 1 (false).

All operators and flags are separate arguments to the *test* utility.

The following primaries are used to construct expression:

–b *file* True if *file* exists and is a block special file.
–c *file* True if *file* exists and is a character special file.
–d *file* True if *file* exists and is a directory.
–e *file* True if *file* exists (regardless of type).
–f *file* True if *file* exists and is a regular file.
–g *file* True if *file* exists and its set group ID flag is set.
–n *string*
True if the length of *string* is nonzero.
–p *file* True if *file* is a named pipe (FIFO).
–r *file* True if *file* exists and is readable.
–s *file* True if *file* exists and has a size greater than zero.
–t *file_descriptor*
True if the file whose file descriptor number is *file_descriptor* (default 1) is open and is associated with a terminal.
–u *file* True if *file* exists and its set user ID flag is set.
–w *file* True if *file* exists and is writeable. True indicates only that the write flag is on. The file is not writeable on a read-only file system even if this test indicates true.
–x *file* True if *file* exists and is executable. True indicates only that the execute flag is on. If *file* is a directory, true indicates that *file* can be searched.
–z *string*
True if the length of *string* is zero.
string True if *string* is not the null string.
s1 **=** *s2* True if the strings *s1* and *s2* are identical.
s1 **!=** *s2* True if the strings *s1* and *s2* are not identical.
n1 **–eq** *n2*
True if the integers *n1* and *n2* are algebraically equal.
n1 **–ne** *n2*
True if the integers *n1* and *n2* are not algebraically equal.
n1 **–gt** *n2*
True if the integer *n1* is algebraically greater than the integer *n2*.
n1 **–ge** *n2*
True if the integer *n1* is algebraically greater than or equal to the integer *n2*.
n1 **–lt** *n2*
True if the integer *n1* is algebraically less than the integer *n2*.

n1 **–le** *n2*
: True if the integer *n1* is algebraically less than or equal to the integer *n2*.

These primaries can be combined with the following operators:

! *expression*
: True if *expression* is false.

expression1 **–a** *expression2*
: True if both *expression1* and *expression2* are true.

expression1 **–o** *expression2*
: True if either *expression1* or *expression2* are true.

(*expression***)**
: True if *expression* is true.

The **–a** operator has higher precedence than the **–o** operator.

DIAGNOSTICS

The *test* utility exits with one of the following values:

0	expression evaluated to true.
1	expression evaluated to false or expression was missing.
>1	An error occurred.

NAME

tftp – trivial file transfer program

SYNOPSIS

`tftp [`*`host`*`]`

DESCRIPTION

Tftp is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case *tftp* uses *host* as the default host for future transfers (see the **connect** command below).

COMMANDS

Once *tftp* is running, it issues the prompt

```
tftp>
```

and recognizes the following commands:

? *command-name ...*
Print help information.

ascii Shorthand for **mode ascii**.

binary Shorthand for **mode binary**.

connect *host-name* [*port*]
Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the **connect** command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the **connect** command; the remote host can be specified as part of the **get** or **put** commands.

get *filename*
get *remotename localname*
get *file1 file2 ... fileN*
Get a file or set of files from the specified *sources. Source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *hosts:filename* to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

mode *transfer-mode* Set the mode for transfers; *transfer-mode* may be one of **ascii** or **binary.** The default is **ascii**.

put *file*
put *localfile remotefile*
put *file1 file2 ... fileN remote-directory*
Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *hosts:filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a machine based on the open standards.

quit Exit *tftp*. An end of file also exits.

rexmt *retransmission-timeout* Set the per-packet retransmission timeout, in

seconds.

status Show current status.

timeout

total-transmission-timeout Set the total transmission timeout, in seconds.

trace Toggle packet tracing.

verbose

Toggle verbose mode.

NOTES

Because there is no user-login or validation within the TFTP protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

The commands **trace**, **verbose**, and **status** are good aids for debugging connection problems

SEE ALSO

tftpd

NAME

tftpd – Internet Trivial File Transfer Protocol server

SYNOPSIS

`tftpd [-ln] [directory ...]`

DESCRIPTION

Tftpd is a server which supports the Internet Trivial File Transfer Protocol (RFC783). The TFTP server operates at the port indicated in the "tftp" service description. The server is normally started by *inetd*.

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Files containing the string "**/../**" are not allowed. Files may be written only if they already exist and are publicly writable. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling tftp service. The server should have the user ID with the lowest possible privilege.

Access to files may be restricted by invoking *tftpd* with a list of directories by including up to 20 pathnames as server program arguments in **$OPENNT_ROOT/etc/inetd.conf.** In this case access is restricted to files whose names are prefixed by the one of the given directories. The given directories are also treated as a search path for relative filename requests.

If no list of directories is specified with *tftp* in the **$OPENNT_ROOT/etc/inetd.conf**file, then all directories are accessible to the remote user. For security reasons, it is highly recommended that at least one directory be listed, and that the **–l** option always.

The options are:

–l Logs all requests using *syslog*.

–n Suppresses negative acknowledgement of requests for nonexistent relative filenames.

SEE ALSO

tftp, *inetd*

NAME

tic - the terminfo entry-description compiler

SYNOPSIS

`tic [-cCINpru] [-v[n]] file`

DESCRIPTION

The command *tic* translates a **terminfo** file from source format into compiled format. The compiled format is necessary for use with the library routines in *ncurses*( ).

The results are normally placed in the system terminfo directory **TERMINFO**. There are two ways to change this behavior.

First, you may override the system default by setting the variable *TERMINFO* in your shell environment to a valid (existing) directory name.

Secondly, if *tic* cannot get access to *TERMINFO* or your **TERMINFO** directory, it looks for the directory **$HOME/.terminfo**; if that directory exists, the entry is placed there.

Libraries that read terminfo entries are expected to check for a **TERMINFO** directory first, look at **$HOME/.terminfo** if TERMINFO is not set, and finally look in *TERMINFO*.

Tic takes the following options:

–c Only check *file* for errors, including syntax problems and bad use links. If you specify **–C** (**–I**) with this option, the code will print warnings about entries which, after use resolution, are more than 1023 (4096) bytes long. Due to a fixed buffer length in older termcap libraries (and a documented limit in terminfo), these entries may cause core dumps.

–v*n* Write verbose output to standard error showing *tic*'s progress. The *n* is optional and specifies the level of detail; it can have a value from 1 to 10 inclusive, with 1 being the least detail and 10 the most. If *n* is not specified, it defaults to 1.

The debug flag levels are as follows:

1 Names of files created and linked
2 Information related to the **use** facility
3 Statistics from the hashing algorithm
5 String-table memory allocations
7 Entries into the string-table
8 List of tokens encountered by scanner
9 All values computed in construction of the hash table

–C Force source translation to termcap format. Capabilities that are not translatable are left in the entry under their terminfo names but commented out with two preceding dots.

–I Force source translation to terminfo format.

–N Disable smart defaults. Normally, when translating from termcap to terminfo, the compiler makes a number of assumptions about the defaults of string capabilities **reset1_string**, **carriage_return**, **cursor_left**, **cursor_down**, **scroll_forward**, **tab**, **newline**, **key_backspace**, **key_left**, and **key_down**, then attempts to use obsolete termcap capabilities to deduce correct values. It also normally suppresses output of obsolete termcap capabilities such as **bs**. This option forces a more literal translation that also preserves the obsolete capabilities.

–r Force entry resolution (so there are no remaining tc capabilities) even when doing translation to termcap format. This may be needed if you are preparing a termcap file for a termcap library (such as GNU termcap up to version 1.3 or BSD termcap up to 4.3BSD) that doesn't handle multiple tc capabilities per entry.

–R*subset* Restrict output to a given subset. This option is for use with archaic versions of terminfo like those on SVr1, Ultrix, or HP/UX that don't support the full set of SVR4/XSI Curses terminfo; and outright broken ports like AIX that have their own extensions incompatible with SVr4/XSI. Available subsets are "SVr1", "Ultrix", "HP", and AIX"; see *terminfo* for details.

The *file* contains one or more **terminfo** terminal descriptions in source format. Each description in the file describes the capabilities of a particular terminal.

All but one of the capabilities recognized by *tic* are documented in *terminfo*. The **use** capability is documented here.

When a terminal entry being compiled contains a field of the form

```
use=entry_name
```

tic reads in the binary from **TERMINFO** to complete the entry. (Entries created from *file* will be used first. If the environment variable *TERMINFO* is set, that directory is searched instead of **TERMINFO**.) *tic* makes the current entry a duplicate of the capabilities for *entry_name*, except for any capabilities explicitly defined for the current entry.

When an entry, e.g., *entry_name_1*, contains a **use=***entry_name_2* field, any canceled capabilities in *entry_name_2* must also appear in *entry_name_1* before **use=** for these capabilities to be canceled in *entry_name_1*.

If the environment variable *TERMINFO* is set, the compiled results are placed there instead of **TERMINFO**.

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes. Terminal names exceeding 14 characters will be truncated to 14 characters and a warning message will be printed.

COMPATIBILITY

There is some evidence that historic *tic* implementations treated description fields with no whitespace in them as additional aliases or shortnames. This *tic* does not do that, but it does warn when description fields may be treated that way and check them for dangerous characters.

EXTENSIONS

Unlike the stock SVr4 *tic* command, this implementation can actually compile termcap sources. In fact, entries in terminfo and termcap syntax can be mixed in a single source file. See *terminfo* for the list of termcap names taken to be equivalent to terminfo names.

The SVr4 manual page are not clear on the resolution rules for **use** capabilities. This implementation of *tic* will find use targets anywhere in the source file, or anywhere in the file tree routed at **TERMINFO** (if *TERMINFO* is defined), or in the user's **$HOME/.terminfo** directory (if it exists), or (finally) anywhere in the system's file tree of compiled entries.

The error messages from this *tic* have the same format as GNU C error messages, and can be parsed by GNU Emacs's compile facility.

FILES

Tic makes use of the following files:

TERMINFO/?/*

Compiled terminal description database.

SEE ALSO

curses

terminfo

NAME

toe – table of (terminfo) entries

SYNOPSIS

`toe [-v[n]] [-huUV] file...`

DESCRIPTION

With no options, *toe* lists all available terminal types by primary name with descriptions. File arguments specify the directories to be scanned; if no such arguments are given, your default terminfo directory is scanned. If you also specify the -h option, a directory header will be issued as each directory is entered.

There are other options intended for use by terminfo file maintainers:

–u *file* says to issue a report on dependencies in the given file. This report condenses the 'use' relation: each line consists of the primary name of a terminal that has use capabilities, followed by a colon, followed by the whitespace-separated primary names of all terminals which occur in those use capabilities, followed by a newline

–U *file* says to issue a report on reverse dependencies in the given file. This report reverses the 'use' relation: each line consists of the primary name of a terminal that occurs in use capabilities, followed by a colon, followed by the whitespace-separated primary names of all terminals which depend on it, followed by a newline.

–v*n* Write (verbose) output to standard error trace information showing *toe*'s progress. The optional integer *n* is a number from 1 to 10, interpreted as for *tic*.

–V

prints out the version of the program in use on standard error and exits.

FILES

terminfo/?/*

Compiled terminal description database.

SEE ALSO

tic,

curses().

NAME

touch – change file access and modification times

SYNOPSIS

`touch [-acm] [-r file] [-t [[CC]YY]MMDDhhmm[.SS]] file ...`

DESCRIPTION

The *touch* utility sets the modification and access times of files to the current time of day. If the file doesn't exist, it is created with default permissions.

The following options are available:

–a Change the access time of the file. The modification time of the file is not changed unless the **–m** flag is also specified.

–c Do not create the file if it does not exist. The *touch* utility does not treat this as an error. No error messages are displayed and the exit value is not affected.

–m Change the modification time of the file. The access time of the file is not changed unless the **–a** flag is also specified.

–r Use the access and modifications times from the specified file instead of the current time of day.

–t Change the access and modification times to the specified time. The argument should be in the form [[*CC*]*YY*]*MMDDhhmm* where each pair of letters represents the following:

CC	The first two digits of the year (the century).
YY	The second two digits of the year. If *YY* is specified but *CC* is not, a *YY* value between 69 and 99 refers to the 20th century (*CC* is 19); a *YY* value between 00 and 68 refers to the 21st century (*CC* is 20).
MM	The month of the year, from 1 to 12.
DD	the day of the month, from 1 to 31.
hh	The hour of the day, from 0 to 23.
mm	The minute of the hour, from 0 to 59.
SS	The second of the minute, from 0 to 61.

If the ''CC'' and ''YY'' letter pairs are not specified, the values default to the current year. If the ''SS'' letter pair is not specified, the value defaults to 0.

DIAGNOSTICS

The *touch* utility exits 0 on success, and >0 if an error occurs.

YEAR 2000 ISSUES

The default behavior of *touch* when the year is specified but the century is not is valid until the year 2068.

NAME

tput – initialize a terminal or query terminfo database

SYNOPSIS

```
tput [-Ttype] capname [parms ... ]
tput [-Ttype] init
tput [-Ttype] reset
tput [-Ttype] longname
tput –S  <<
```

DESCRIPTION

The *tput* utility uses the **terminfo** database to make the values of terminal-dependent capabilities and information available to the shell (see *sh*), to initialize or reset the terminal, or return the long name of the requested terminal type. *tput* outputs a string if the attribute (*cap*ability *name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, *tput* simply sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code [**$?**, see *sh*] to be sure it is **0**. (See the "DIAGNOSTICS" and "DIAGNOSTICS" sections.) For a complete list of capabilities and the *capname* associated with each, see *terminfo*. *Tput* accepts the following options:

–T*type* specifies the *type* of terminal. Normally this option is unnecessary, because the default is taken from the environment variable *TERM*. If **–T** is specified, then *tput* ignores the shell variables *LINES* and *COLUMNS* and does not query the operating system for the actual screen size.

–S allows more than one capability per invocation of *tput*. The capabilities must be passed to *tput* from the standard input instead of from the command line (see example). Only one *capname* is allowed per line. The **–S** option changes the meaning of the **0** and **1** boolean and string exit codes (see the "DIAGNOSTICS" section).

capname Names the attribute from the **terminfo** database.

parms If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.

init If the **terminfo** database is present and an entry for the user's terminal exists (see **–T**), the following will occur:

1. if present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**),
2. any delays (e.g., newline) specified in the entry will be set in the tty driver,
3. tabs expansion will be turned on or off according to the specification in the entry, and
4. if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset Instead of putting out initialization strings, the terminal's reset strings will be output if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname
If the **terminfo** database is present and an entry for the user's terminal exists (see **–T**), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the **terminfo** database [see *term*].

EXAMPLES

Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command should be included in everyone's **$HOME/.profile** after the environmental variable *TERM* has been exported.

```
tput init
```

Reset an AT&T 5620 terminal, overriding the type of terminal in the environmental variable **TERM**.

```
tput -T5620 reset
```

Send the sequence to move the cursor to row **0**, column **0** (the upper left corner of the screen, usually known as the "home" cursor position).

```
tput cup 0 0
```

Echo the clear-screen sequence for the current terminal.

```
tput clear
```

Print the number of columns for the current terminal.

```
tput cols
```

Print the number of columns for the 450 terminal.

```
tput -T450 cols
```

Set the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end standout mode sequence, for the current terminal. This might be followed by a prompt: **echo** "**${bold}Please type in your name: ${offbold}\c**"

```
bold='tput smso' offbold='tput rmso'
```

Set exit code to indicate if the current terminal is a hardcopy terminal.

```
tput hc
```

Send the sequence to move the cursor to row 23, column 4.

```
tput cup 23 4
```

Print the long name from the **terminfo** database for the type of terminal specified in the environmental variable *TERM*.

```
tput longname
```

This example shows *tput* processing several capabilities in one invocation. This example clears the screen, moves the cursor to position 10, 10 and turns on bold (extra bright) mode. The list is terminated by an exclamation mark (**!**) on a line by itself.

```
tput -S <<!
> clear
> cup 10 10
> bold
> !
```

FILES

Tput makes use of the the following files:

TERMINFO
Compiled terminal description database.

$OPENNT_ROOT/usr/include/curses.h
The *curses*( ) header file

$OPENNT_ROOT/usr/include/term.h
The *terminfo* header file

/usr/lib/tabset/*
Tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the "Tabs and Initialization" section of *terminfo*

DIAGNOSTICS

The exit code depends upon the type of the capability, here given as *capname.*

If *capname* is of type boolean, a value of **0** is set for TRUE and **1** for FALSE unless the **–S** option is used.

If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname* is of type boolean or string and the **–S** option is used, a value of **0** is returned to indicate that all lines were successful. No indication of which line failed can be given so exit code **1** will never appear. Exit codes **2**, **3**, and **4** retain their usual interpretation.

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type.* To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of **-1** means that *capname* is not defined for this terminal *type.*

Any other exit code indicates an error.

Tput prints the following error messages and sets the corresponding exit codes.

Exit Code	Error Message
0	(*capname* is a numeric variable that is not specified in the terminfo(5) database for this terminal type, e.g. tput -T450 lines and **tput -T2621 xmc**)
1	no error message is printed, see the “EXIT CODES” section.
2	usage error
3	unknown terminal *type* or no **terminfo** database

4 unknown **terminfo** capability *capname*

PORTABILITY

The **longname** and **–S** options, and the parameter-substitution features used in the **cup** example, are not supported in BSD curses or in AT&T/USL curses before SVr4.

SEE ALSO

clear, *stty*

terminfo

NAME

tr – translate characters

SYNOPSIS

```
tr [-cs] string1 string2
tr [-c] -d string1
tr [-c] -s string1
tr [-c] -ds string1 string2
```

DESCRIPTION

The *tr* utility copies the standard input to the standard output with substitution or deletion of selected characters.

The following options are available:

–c Complements the set of characters in *string1*, that is "–c ab" includes every character except for "a" and "b".

–d The **–d** option causes characters to be deleted from the input.

–s The **–s** option squeezes multiple occurrences of the characters listed in the last operand (either *string1* or *string2*) in the input into a single instance of the character. This occurs after all deletion and translation is completed.

In the first synopsis form, the characters in *string1* are translated into the characters in *string2*. The first character in *string1* is translated into the first character in *string2* and so on. If *string1* is longer than *string2*, the last character found in *string2* is duplicated until *string1* is exhausted.

In the second synopsis form, the characters in *string1* are deleted from the input.

In the third synopsis form, the characters in *string1* are compressed as described for the **–s** option.

In the fourth synopsis form, the characters in *string1* are deleted from the input, and the characters in *string2* are compressed as described for the **–s** option.

The following conventions can be used in *string1* and *string2* to specify sets of characters:

c Any character *c* not described by one of the following conventions represents itself.

octal A backslash followed by 1, 2 or 3 octal digits represents a character with that encoded value. To follow an octal sequence with a digit as a character, left zero-pad the octal sequence to the full 3 octal digits.

character

A backslash followed by certain special characters maps to special values.

\a alert character

\b	backspace
\f	form-feed
\n	newline
\r	carriage return
\t	tab
\v	vertical tab

A backslash followed by any other character maps to that character.

c-c Represents the range of characters between the range endpoints, inclusively.

[:*class*:] Represents all characters belonging to the defined character class. Class names are:

alnum	alphanumeric characters
alpha	alphabetic characters
cntrl	control characters
digit	numeric characters
graph	graphic characters
lower	lower-case alphabetic characters
print	printable characters
punct	punctuation characters
space	space characters
upper	upper-case characters
xdigit	hexadecimal characters

With the exception of the “upper” and “lower” classes, characters in the classes are in unspecified order. In the “upper” and “lower” classes, characters are entered in ascending order.

[=*equiv*=] Represents all characters or collating (sorting) elements belonging to the same equivalence class as *equiv*. If there is a secondary ordering within the equivalence class, the characters are ordered in ascending sequence. Otherwise, they are ordered after their encoded values. An example of an equivalence class might be “c” and “ch” in Spanish; English has no equivalence classes.

[*c***n*] Represents *n* repeated occurrences of the character represented by *c*. This expression is only valid when it occurs in *string2*. If *n* is omitted or is zero, it is be interpreted as large enough to extend *string2* sequence to the length of *string1*. If *n* has a leading zero, it is interpreted as an octal value, otherwise, it’s interpreted as a decimal value.

The *tr* utility exits 0 on success, and >0 if an error occurs.

EXAMPLES

The following examples are shown as given to the shell:

Create a list of the words in file1, one per line, where a word is taken to be a maximal string of letters.

```
tr -cs "[:alpha:]" "\n" < file1
```

Translate the contents of file1 to upper-case.

```
tr "[:lower:]" "[:upper:]" < file1
```

Strip out non-printable characters from file1.

```
tr -cd "[:print:]" < file1
```

Perform (or undo) a “rot13” transformation on text in file1:

```
tr "[A-Ma-mN-Zn-z]" "[N-Zn-zA-Ma-m]" < file1
```

COMPATIBILITY

System V has historically implemented character ranges using the syntax “[c-c]” instead of the “c-c” used by historic BSD implementations and standardized by POSIX. System V shell scripts should work under this implementation as long as the range is intended to map in another range, i.e. the command “tr [a-z] [A-Z]” will work as it will map the “[” character in *string1* to the “[” character in *string2*. However, if the shell script is deleting or squeezing characters as in the command “tr -d [a-z]”, the characters “[” and “]” will be included in the deletion or compression list which would not have happened under an historic System V implementation. Additionally, any scripts that depended on the sequence “a-z” to represent the three characters “a”, “-” and “z” will have to be rewritten as “a\-z”.

The *tr* utility has historically not permitted the manipulation of NUL bytes in its input and, additionally, stripped NUL’s from its input stream. This implementation has removed this behavior as a bug.

The *tr* utility has historically been extremely forgiving of syntax errors, for example, the **–c** and **–s** options were ignored unless two strings were specified. This implementation will not permit illegal syntax.

NAME

true – return true value

SYNOPSIS

```
true
```

DESCRIPTION

True is normally used in a shell script. *True* tests for the appropriate status "false" before running (or failing to run) a list of commands.

DIAGNOSTICS

The *true* utility always returns with exit code zero.

SEE ALSO

false, *sh*

NAME

tset – terminal initialization

SYNOPSIS

```
tset [-IQrs] [-] [-e ch] [-i ch] [-k ch]
     [-m mapping] [terminal]

reset [-IQrs] [-] [-e ch] [-i ch] [-k ch]
      [-m mapping] [terminal]
```

DESCRIPTION

Tset initializes terminals. *Tset* first determines the type of terminal that you are using. This determination is done as follows, using the first terminal type found.

1. The *terminal* argument specified on the command line.
2. The value of the *TERM* environmental variable.
3. The default terminal type, ‘‘unknown’’.

If the terminal type was not specified on the command-line, the **–m** option mappings are then applied (see below for more information). Then, if the terminal type begins with a question mark (‘‘?’’), the user is prompted for confirmation of the terminal type. An empty response confirms the type, or, another type can be entered to specify a new type. Once the terminal type has been determined, the terminfo entry for the terminal is retrieved. If no terminfo entry is found for the type, the user is prompted for another terminal type.

Once the terminfo entry is retrieved, the window size, backspace, interrupt and line kill characters (among many other things) are set and the terminal and tab initialization strings are sent to the standard error output. Finally, if the erase, interrupt and line kill characters have changed, or are not set to their default values, their values are displayed to the standard error output.

The *tset* command can also be installed as *reset*. When invoked as *reset*, *tset* sets cooked and echo modes, turns off cbreak and raw modes, turns on newline translation and resets any unset special characters to their default values before doing the terminal initialization described above. This is useful after a program dies leaving a terminal in an abnormal state. Note, you may have to type

```
<LF>reset<LF>A
```

(the line-feed character is normally control-J) to get the terminal to work, as carriage-return may no longer work in the abnormal state. Also, the terminal will often not echo the command.

The options are as follows:

– The terminal type is displayed to the standard output, and the terminal is not initialized in any way.

–e *ch* Set the erase character to *ch*.

–I Do not send the terminal or tab initialization strings to the terminal.

–i *ch* Set the interrupt character to *ch*.

–k *ch* Set the line kill character to *ch*.

–m Specify a mapping from a port type to a terminal. See below for more information.

–Q Don't display any values for the erase, interrupt and line kill characters.

–r Print the terminal type to the standard error output.

–s Print the sequence of shell commands to initialize the environment variable *TERM* to the standard output. See the section below on setting the environment for details.

The arguments for the **–e**, **–i**, and **–k** options may either be entered as actual characters or by using the 'hat' notation, i.e. control-h may be specified as ''^H'' or ''^h''.

SETTING THE ENVIRONMENT

It is often desirable to enter the terminal type and information about the terminal's capabilities into the shell's environment. This is done using the **–s** option.

When the **–s** option is specified, the commands to enter the information into the shell's environment are written to the standard output. If the *SHELL* environmental variable ends in ''csh'', the commands are for *csh*, otherwise, they are for *sh*. Note, the *csh* commands set and unset the shell variable *noglob*, leaving it unset. The following line in the **.login** or **.profile** files will initialize the environment correctly:

```
eval `tset -s options ... `
```

TERMINAL TYPE MAPPING

When the terminal is not hardwired into the system (or the current system information is incorrect) the terminal type derived from the *TERM* environmental variable is often something generic like **network**, **dialup**, or **unknown**. When *tset* is used in a startup script it is often desirable to provide information about the type of terminal used on such ports.

The purpose of the **–m** option is to map from some set of conditions to a terminal type, that is, to tell *tset* "If I'm on this port at a particular speed, guess that I'm on that kind of terminal".

The argument to the **–m** option consists of an optional port type, an optional operator, an optional baud rate specification, an optional colon (':') character and a terminal type. The port type is a string (delimited by either the operator or the colon character). The operator may be any combination of '>', '<', '@', and '!'; '>' means greater than, '<' means less than, '@' means equal to and '!' inverts the sense of the test. The baud rate is specified as a number and is compared with the speed of the standard error output (which should be the control terminal). The terminal type is a string.

If the terminal type is not specified on the command line, the **–m** mappings are applied to the terminal type. If the port type and baud rate match the mapping, the terminal type specified in the mapping replaces the current type. If more than one mapping is specified, the first applicable mapping is used.

For example, consider the following mapping: **dialup>9600:vt100**. The port type is dialup , the operator is >, the baud rate specification is 9600, and the terminal type is vt100. The result of this mapping is to specify that if the terminal type is **dialup**, and the baud rate is greater than 9600 baud, a terminal type of **vt100** will be used.

If no baud rate is specified, the terminal type will match any baud rate. If no port type is specified, the terminal type will match any port type. For example, **-m dialup:vt100 -m :?xterm** will cause any dialup port, regardless of baud rate, to match the terminal type vt100, and any non-dialup port type to match the terminal type ?xterm. Note, because of the leading question mark, the user will be queried on a default port as to whether they are actually using an xterm terminal.

No whitespace characters are permitted in the **–m** option argument. Also, to avoid problems with metacharacters, it is suggested that the entire **–m** option argument be placed within single quote characters, and that *csh* users insert a backslash character (‘‘\’’) before any exclamation marks (‘‘!’’).

NOTES

The **ncurses** implementation was lightly adapted from the 4.4BSD sources for a terminfo environment by Eric S. Raymond <esr@snark.thyrsus.com>.

The *tset* utility has been provided for backward-compatibility with BSD environments. This implementation behaves like 4.4BSD *tset*, with a few exceptions specified here.

The **–S** option of BSD *tset* no longer works; it prints an error message to stderr and dies. The **–s** option only sets *TERM*, not *TERMCAP*. Both these changes are because the *TERMCAP* variable is no longer supported under terminfo-based **ncurses**, which makes **tset -S** useless (we made it die noisily rather than silently induce lossage).

There was an undocumented 4.4BSD feature that invoking *tset* via a link named ‘TSET‘ (or via any other name beginning with an upper-case letter) set the terminal to use upper-case only. This feature has been omitted.

The **–A**, **–E**, **–h**, **–u** and **–v** options were deleted from the *tset* utility in 4.4BSD. None of them were documented in 4.3BSD and all are of limited utility at best. The **–a**, **–d**, and **–p** options are similarly not documented or useful, but were retained as they appear to be in widespread use. It is strongly recommended that any usage of these three options be changed to use the m**–option**instead. **–n** option remains, but has no effect. The **–adnp** options are therefore omitted from the usage summary above.

It is still permissible to specify the **–e**, **–i**, and **–k** options without arguments, although it is strongly recommended that such usage be fixed to explicitly specify the character.

As of 4.4BSD, executing *tset* as *reset* no longer implies the **–Q** option. Also, the interaction between the - option and the *terminal* argument in some historic

implementations of *tset* has been removed.

ENVIRONMENT

The *tset* command uses the *SHELL* and *TERM* environment variables.

FILES

$OPENNT_ROOT/usr/share/terminfo
Terminal capability database.

NOTES

The default installation does not install the *tset* command as *reset*. Use the *ln* command (if INTERIX is installed on an NTFS partition) or the *cp* command to create *reset*.

SEE ALSO

csh, *sh*, *stty*

tty(4)

termcap

NAME

tsort – topological sort of a directed graph

SYNOPSIS

`tsort [-l] [file]`

DESCRIPTION

Tsort takes a list of pairs of node names representing directed arcs in a graph and prints the nodes in topological order on standard output. Input is taken from the named *file*, or from standard input if no file is given.

Node names in the input are separated by white space and there must be an even number of node pairs.

Presence of a node in a graph can be represented by an arc from the node to itself. This is useful when a node is not connected to any other nodes.

If the graph contains a cycle (and therefore cannot be properly sorted), one of the arcs in the cycle is ignored and the sort continues. Cycles are reported on standard error.

The options are as follows:

–l Search for and display the longest cycle. Can take a very long time.

DIAGNOSTICS

The *tsort* command exits with status 0 on success, and a status >0 on failure.

NOTES

This *tsort* command and manual page are derived from sources contributed to Berkeley by Michael Rendell of Memorial University of Newfoundland.

SEE ALSO

ar

NAME

tty – return user's terminal name

SYNOPSIS

```
tty [-s]
```

DESCRIPTION

The *tty* utility writes to standard output the name of the terminal attached to standard input. The name is the string returned by *ttyname*( ). If the standard input is not a terminal, the message "not a tty" is written. The options are as follows:

–s Don't write the terminal name; only the exit status is affected when this option is specified.

The **–s** option is deprecated in favor of:

```
test -t 0
```

DIAGNOSTICS

Tty exits 0 if the standard input is a terminal, 1 if the standard input is not a terminal, and >1 if an error occurs.

SEE ALSO

test

NAME

twm - Tab Window Manager for the X Window System

SYNOPSIS

`twm [–display` *`dpy`*`] [-s] [-f` *`initfile`*`] [-v]`

DESCRIPTION

Twm is a window manager for the X Window System. It provides titlebars, shaped windows, several forms of icon management, user-defined macro functions, click-to-type and pointer-driven keyboard focus, and user-specified key and pointer button bindings.

This program is usually started by the user's session manager or startup script. When used from *xdm* or *xinit* without a session manager, *twm* is frequently executed in the foreground as the last client. When run this way, exiting *twm* causes the session to be terminated (i.e. logged out).

By default, application windows are surrounded by a ''frame'' with a titlebar at the top and a special border around the window. The titlebar contains the window's name, a rectangle that is lit when the window is receiving keyboard input, and function boxes known as ''titlebuttons'' at the left and right edges of the titlebar.

Pressing pointer Button1 (usually the left-most button unless it has been changed with *xmodmap*) on a titlebutton will invoke the function associated with the button. In the default interface, windows are iconified by clicking (pressing and then immediately releasing) the left titlebutton (which looks like a Dot). Conversely, windows are deiconified by clicking in the associated icon or entry in the icon manager (see description of the variable **ShowIconManager** and of the function **f.showiconmgr**).

Windows are resized by pressing the right titlebutton (which resembles a group of nested squares), dragging the pointer over edge that is to be moved, and releasing the pointer when the outline of the window is the desired size. Similarly, windows are moved by pressing in the title or highlight region, dragging a window outline to the new location, and then releasing when the outline is in the desired position. Just clicking in the title or highlight region raises the window without moving it.

When new windows are created, *twm* will honor any size and location information requested by the user (usually through **–geometry** command line argument or resources for the individual applications). Otherwise, an outline of the window's default size, its titlebar, and lines dividing the window into a 3x3 grid that track the pointer are displayed. Clicking pointer Button1 will position the window at the current position and give it the default size. Pressing pointer Button2 (usually the middle pointer button) and dragging the outline will give the window its current position but allow the sides to be resized as described above. Clicking pointer Button3 (usually the right pointer button) will give the window its current position but attempt to make it long enough to touch the bottom the screen.

OPTIONS

Twm accepts the following command line options:

–display *dpy*
This option specifies the X server to use.

–s This option indicates that only the default screen (as specified by **–display** or by the *DISPLAY* environment variable) should be managed. By default, *twm* will attempt to manage all screens on the display.

–f *filename*
This option specifies the name of the startup file to use. By default, *twm* will look in the user's home directory for files named **.twmrc.***num* (where *num* is a screen number) or **.twmrc**.

–v This option indicates that *twm* should print error messages whenever an unexpected X Error event is received. This can be useful when debugging applications but can be distracting in regular use.

CUSTOMIZATION

Much of *twm*'s appearance and behavior can be controlled by providing a startup file in one of the following locations (searched in order for each screen being managed when *twm* begins):

$HOME/.twmrc.*screennumber*
The *screennumber* is a small positive number (e.g. 0, 1, etc.) representing the screen number (e.g. the last number in the *DISPLAY* environment variable *host*:*displaynum*.*screennum*) that would be used to contact that screen of the display. This is intended for displays with multiple screens of differing visual types.

$HOME/.twmrc
This is the usual name for an individual user's startup file.

$OPENNT_ROOT/usr/lib/X11/twm/system.twmrc
If neither of the preceding files are found, *twm* will look in this file for a default configuration. This is often tailored by the site administrator to provide convenient menus or familiar bindings for novice users.

If no startup files are found, *twm* will use the built-in defaults described above. The only resource used by *twm* is *bitmapFilePath* for a colon-separated list of directories to search when looking for bitmap files (for more information, see the *Athena Widgets* manual and *xrdb*.)

Twm startup files are logically broken up into three types of specifications: *Variables*, *Bindings*, *Menus*. The *Variables* section must come first and is used to describe the fonts, colors, cursors, border widths, icon and window placement, highlighting, autoraising, layout of titles, warping, use of the icon manager. The *Bindings* section usually comes second and is used to specify the functions that should be to be invoked when keyboard and pointer buttons are pressed in windows, icons, titles, and frames. The *Menus* section gives any user-defined menus (containing functions to be invoked or commands to be executed).

Variable names and keywords are case-insensitive. Strings must be surrounded by double quote characters (e.g. "blue") and are case-sensitive. A pound sign (#) outside of a string causes the remainder of the line in which the character appears to be treated as a comment.

VARIABLES

Many of the aspects of *twm*'s user interface are controlled by variables that may be set in the user's startup file. Some of the options are enabled or disabled simply by the presence of a particular keyword. Other options require keywords, numbers, strings, or lists of all of these.

Lists are surrounded by braces and are usually separated by whitespace or a newline. For example:

```
AutoRaise { "emacs" "XTerm" "Xmh" }
```
or
```
AutoRaise
{
        "emacs"
        "XTerm"
        "Xmh"
}
```

When a variable containing a list of strings representing windows is searched (e.g. to determine whether or not to enable autoraise as shown above), a string must be an exact, case-sensitive match to the window's name name (given by the WM_NAME window property), resource name or class name (both given by the WM_CLASS window property). The preceding example would enable autoraise on windows named ''emacs'' as well as any *xterm* (since they are of class ''XTerm'') or xmh windows (which are of class ''Xmh'').

String arguments that are interpreted as filenames (see the **Pixmaps**, **Cursors**, and **IconDirectory** below) will prepend the user's directory (specified by the *HOME* environment variable) if the first character is a tilde (~). If, instead, the first character is a colon (:), the name is assumed to refer to one of the internal bitmaps that are used to create the default titlebars symbols: **:xlogo** or **:iconify** (both refer to the X used for the iconify button), **:resize** (the nested squares used by the resize button), and **:question** (the question mark used for non-existent bitmap files).

The following variables may be specified at the top of a *twm* startup file. Lists of Window name prefix strings are indicated by *win-list*. Optional arguments are shown in square brackets:

AutoRaise { *win-list* }
: This variable specifies a list of windows that should automatically be raised whenever the pointer enters the window. This action can be interactively enabled or disabled on individual windows using the function **f.autoraise**.

AutoRelativeResize
: This variable indicates that dragging out a window size (either when initially sizing the window with pointer Button2 or when resizing it) should not wait until the pointer has crossed the window edges. Instead, moving the pointer automatically causes the nearest edge or edges to move by the same amount. This allows the resizing of windows that extend off the edge of the screen. If the pointer is in the center of the window, or if the resize is begun by pressing a titlebutton, *twm* will still wait for the pointer to cross a window edge

(to prevent accidents). This option is particularly useful for people who like the press-drag-release method of sweeping out window sizes.

BorderColor *string* [{ *wincolorlist* }]
This variable specifies the default color of the border to be placed around all non-iconified windows, and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* specifies a list of window and color name pairs for specifying particular border colors for different types of windows. For example:

```
BorderColor "gray50"
{
        "XTerm"       "red"
        "xmh"  "green"
}
```

The default is "black".

BorderTileBackground *string* [{ *wincolorlist* }]
This variable specifies the default background color in the gray pattern used in unhighlighted borders (only if **NoHighlight** hasn't been set), and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* allows per-window colors to be specified. The default is "white".

BorderTileForeground *string* [{ *wincolorlist* }]
This variable specifies the default foreground color in the gray pattern used in unhighlighted borders (only if **NoHighlight** hasn't been set), and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* allows per-window colors to be specified. The default is "black".

BorderWidth *pixels*
This variable specifies the width in pixels of the border surrounding all client window frames if **ClientBorderWidth** has not been specified. This value is also used to set the border size of windows created by *twm* (such as the icon manager). The default is 2.

ButtonIndent *pixels*
This variable specifies the amount by which titlebuttons should be indented on all sides. Positive values cause the buttons to be smaller than the window text and highlight area so that they stand out. Setting this and the **TitleButtonBorderWidth** variables to 0 makes titlebuttons be as tall and wide as possible. The default is 1.

ClientBorderWidth
This variable indicates that border width of a window's frame should be set to the initial border width of the window, rather than to the value of **BorderWidth**.

Color { *colors-list* }
This variable specifies a list of color assignments to be made if the default display is capable of displaying more than simple black and white. The *colors-list* is made up of the following color variables and their values: **DefaultBackground**, **DefaultForeground**,

MenuBackground, **MenuForeground**, **MenuTitleBackground**, **MenuTitleForeground**, and **MenuShadowColor**.

The following color variables may also be given a list of window and color name pairs to allow per-window colors to be specified (see **BorderColor** for details): **BorderColor**, **IconManagerHighlight**, **BorderTitleBackground**, **BorderTitleForeground**, **TitleBackground**, **TitleForeground**, **IconBackground**, **IconForeground**, **IconBorderColor**, **IconManagerBackground**, and **IconManagerForeground**. For example:

```
Color
{
        MenuBackground      "gray50"
        MenuForeground      "blue"
        BorderColor   "red" { "XTerm" "yellow" }
        TitleForeground     "yellow"
        TitleBackground     "blue"
}
```

All of these color variables may also be specified for the **Monochrome** variable, allowing the same initialization file to be used on both color and monochrome displays.

ConstrainedMoveTime *milliseconds*

This variable specifies the length of time between button clicks needed to begin a constrained move operation. Double clicking within this amount of time when invoking **f.move** will cause the window only be moved in a horizontal or vertical direction. Setting this value to 0 will disable constrained moves. The default is 400 milliseconds.

Cursors { *cursor-list* }

This variable specifies the glyphs that *twm* should use for various pointer cursors. Each cursor may be defined either from the **cursor** font or from two bitmap files. Shapes from the **cursor** font may be specified directly as:

cursorname "*string*"

where *cursorname* is one of the cursor names listed below, and *string* is the name of a glyph as found in the file /usr/include/X11/cursorfont.h (without the ''XC_'' prefix). If the cursor is to be defined from bitmap files, the following syntax is used instead:

cursorname "*image*" "*mask*"

The *image* and *mask* strings specify the names of files containing the glyph image and mask in *bitmap(1)* form. The bitmap files are located in the same manner as icon bitmap files. The following example shows the default cursor definitions:

```
Cursors
{
        Frame         "top_left_arrow"
```

```
        Title       "top_left_arrow"
        Icon        "top_left_arrow"
        IconMgr     "top_left_arrow"
        Move        "fleur"
        Resize      "fleur"
        Menu        "sb_left_arrow"
        Button      "hand2"
        Wait        "watch"
        Select      "dot"
        Destroy     "pirate"
}
```

DecorateTransients
This variable indicates that transient windows (those containing a WM_TRANSIENT_FOR property) should have titlebars. By default, transients are not reparented.

DefaultBackground *string*
This variable specifies the background color to be used for sizing and information windows. The default is "white".

DefaultForeground *string*
This variable specifies the foreground color to be used for sizing and information windows. The default is "black".

DontIconifyByUnmapping { *win-list* }
This variable specifies a list of windows that should not be iconified by simply unmapping the window (as would be the case if **IconifyByUnmapping** had been set). This is frequently used to force some windows to be treated as icons while other windows are handled by the icon manager.

DontMoveOff
This variable indicates that windows should not be allowed to be moved off the screen. It can be overridden by the **f.forcemove** function.

DontSqueezeTitle [{ *win-list* }]
This variable indicates that titlebars should not be squeezed to their minimum size as described under **SqueezeTitle** below. If the optional window list is supplied, only those windows will be prevented from being squeezed.

ForceIcons
This variable indicates that icon pixmaps specified in the **Icons** variable should override any client-supplied pixmaps.

FramePadding *pixels*
This variable specifies the distance between the titlebar decorations (the button and text) and the window frame. The default is 2 pixels.

IconBackground *string* [{ *win-list* }]
This variable specifies the background color of icons, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description

of the *win-list*. The default is "white".

IconBorderColor *string* [{ *win-list* }]
This variable specifies the color of the border used for icon windows, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconBorderWidth *pixels*
This variable specifies the width in pixels of the border surrounding icon windows. The default is 2.

IconDirectory *string*
This variable specifies the directory that should be searched if if a bitmap file cannot be found in any of the directories in the **bitmapFilePath** resource.

IconFont *string*
This variable specifies the font to be used to display icon names within icons. The default is "variable".

IconForeground *string* [{ *win-list* }]
This variable specifies the foreground color to be used when displaying icons, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconifyByUnmapping [{ *win-list* }]
This variable indicates that windows should be iconified by being unmapped without trying to map any icons. This assumes that the user is will remap the window through the icon manager, the **f.warpto** function, or the *TwmWindows* menu. If the optional *win-list* is provided, only those windows will be iconified by simply unmapping. Windows that have both this and the **IconManagerDontShow** options set may not be accessible if no binding to the *TwmWindows* menu is set in the user's startup file.

IconManagerBackground *string* [{ *win-list* }]
This variable specifies the background color to use for icon manager entries, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "white".

IconManagerDontShow [{ *win-list* }]
This variable indicates that the icon manager should not display any windows. If the optional *win-list* is given, only those windows will not be displayed. This variable is used to prevent windows that are rarely iconified (such as *xclock* or *xload*) from taking up space in the icon manager.

IconManagerFont *string*
This variable specifies the font to be used when displaying icon

manager entries. The default is "variable".

IconManagerForeground *string* [{ *win-list* }]
This variable specifies the foreground color to be used when displaying icon manager entries, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconManagerGeometry *string* [*columns*]
This variable specifies the geometry of the icon manager window. The *string* argument is standard geometry specification that indicates the initial full size of the icon manager. The icon manager window is then broken into *columns* pieces and scaled according to the number of entries in the icon manager. Extra entries are wrapped to form additional rows. The default number of columns is 1.

IconManagerHighlight *string* [{ *win-list* }]
This variable specifies the border color to be used when highlighting the icon manager entry that currently has the focus, and can only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconManagers { *iconmgr-list* }
This variable specifies a list of icon managers to create. Each item in the *iconmgr-list* has the following format:

```
"winname" ["iconname"]     "geometry" columns
```

where *winname* is the name of the windows that should be put into this icon manager, *iconname* is the name of that icon manager window's icon, *geometry* is a standard geometry specification, and *columns* is the number of columns in this icon manager as described in **IconManagerGeometry**. For example:

```
IconManagers
{
        "XTerm"         "=300x5+800+5"      5
        "myhost"        "=400x5+100+5"      2
}
```

Clients whose name or class is ''XTerm'' will have an entry created in the ''XTerm'' icon manager. Clients whose name was ''myhost'' would be put into the ''myhost'' icon manager.

IconManagerShow { *win-list* }
This variable specifies a list of windows that should appear in the icon manager. When used in conjunction with the **IconManagerDontShow** variable, only the windows in this list will be shown in the icon manager.

IconRegion *geomstring vgrav hgrav gridwidth gridheight*
This variable specifies an area on the root window in which icons are

placed if no specific icon location is provided by the client. The *geomstring* is a quoted string containing a standard geometry specification. If more than one **IconRegion** lines are given, icons will be put into the succeeding icon regions when the first is full. The *vgrav* argument should be either **North** or **South** and control and is used to control whether icons are first filled in from the top or bottom of the icon region. Similarly, the *hgrav* argument should be either **East** or **West** and is used to control whether icons should be filled in from left from the right. Icons are laid out within the region in a grid with cells *gridwidth* pixels wide and *gridheight* pixels high.

Icons { *win-list* }

This variable specifies a list of window names and the bitmap filenames that should be used as their icons. For example:

```
Icons
{
        "XTerm"         "xterm.icon"
        "xfd"           "xfd_icon"
}
```

Windows that match ''XTerm'' and would not be iconified by unmapping, and would try to use the icon bitmap in the file ''xterm.icon''. If **ForceIcons** is specified, this bitmap will be used even if the client has requested its own icon pixmap.

InterpolateMenuColors

This variable indicates that menu entry colors should be interpolated between entry specified colors. In the example below:

```
Menu "mymenu"
{
        "Title"             ("black":"red")     f.title
        "entry1"                                f.nop
        "entry2"                                f.nop
        "entry3"        ("white":"green")       f.nop
        "entry4"                                f.nop
        "entry5"        ("red":"white")                 f.nop
}
```

the foreground colors for ''entry1'' and ''entry2'' will be interpolated between black and white, and the background colors between red and green. Similarly, the foreground for ''entry4'' will be half-way between white and red, and the background will be half-way between green and white.

MakeTitle { *win-list* }

This variable specifies a list of windows on which a titlebar should be placed and is used to request titles on specific windows when **NoTitle** has been set.

MaxWindowSize *string*

This variable specifies a geometry in which the width and height give the maximum size for a given window. This is typically used to restrict windows to the size of the screen. The default is

"30000x30000".

MenuBackground *string*
This variable specifies the background color used for menus, and can only be specified inside of a **Color** or **Monochrome** list. The default is "white".

MenuFont *string*
This variable specifies the font to use when displaying menus. The default is "variable".

MenuForeground *string*
This variable specifies the foreground color used for menus, and can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

MenuShadowColor *string*
This variable specifies the color of the shadow behind pull-down menus and can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

MenuTitleBackground *string*
This variable specifies the background color for **f.title** entries in menus, and can only be specified inside of a **Color** or **Monochrome** list. The default is "white".

MenuTitleForeground *string*
This variable specifies the foreground color for **f.title** entries in menus and can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

Monochrome { *colors* }
This variable specifies a list of color assignments that should be made if the screen has a depth of 1. See the description of **Colors**.

MoveDelta *pixels*
This variable specifies the number of pixels the pointer must move before the **f.move** function starts working. Also see the **f.deltastop** function. The default is zero pixels.

NoBackingStore
This variable indicates that *twm*'s menus should not request backing store to minimize repainting of menus. This is typically used with servers that can repaint faster than they can handle backing store.

NoCaseSensitive
This variable indicates that case should be ignored when sorting icon names in an icon manager. This option is typically used with applications that capitalize the first letter of their icon name.

NoDefaults
This variable indicates that *twm* should not supply the default titlebuttons and bindings. This option should only be used if the startup file contains a completely new set of bindings and definitions.

NoGrabServer
This variable indicates that *twm* should not grab the server when

popping up menus and moving opaque windows.

NoHighlight [{ *win-list* }]
This variable indicates that borders should not be highlighted to track the location of the pointer. If the optional *win-list* is given, highlighting will only be disabled for those windows. When the border is highlighted, it will be drawn in the current **BorderColor**. When the border is not highlighted, it will be stippled with an gray pattern using the current **BorderTileForeground** and **BorderTileBackground** colors.

NoIconManagers
This variable indicates that no icon manager should be created.

NoMenuShadows
This variable indicates that menus should not have drop shadows drawn behind them. This is typically used with slower servers since it speeds up menu drawing at the expense of making the menu slightly harder to read.

NoRaiseOnDeiconify
This variable indicates that windows that are deiconified should not be raised.

NoRaiseOnMove
This variable indicates that windows should not be raised when moved. This is typically used to allow windows to slide underneath each other.

NoRaiseOnResize
This variable indicates that windows should not be raised when resized. This is typically used to allow windows to be resized underneath each other.

NoRaiseOnWarp
This variable indicates that windows should not be raised when the pointer is warped into them with the **f.warpto** function. If this option is set, warping to an occluded window may result in the pointer ending up in the occluding window instead the desired window (which causes unexpected behavior with **f.warpring**).

NoSaveUnders
This variable indicates that menus should not request save-unders to minimize window repainting following menu selection. It is typically used with displays that can repaint faster than they can handle save-unders.

NoStackMode [{ *win-list* }]
This variable indicates that client window requests to change stacking order should be ignored. If the optional *win-list* is given, only requests on those windows will be ignored. This is typically used to prevent applications from relentlessly popping themselves to the front of the window stack.

NoTitle [{ *win-list* }]
This variable indicates that windows should not have titlebars. If the

optional *win-list* is given, only those windows will not have titlebars. **MakeTitle** may be used with this option to force titlebars to be put on specific windows.

NoTitleFocus
This variable indicates that *twm* should not set keyboard input focus to each window as it is entered. Normally, *twm* sets the focus so that focus and key events from the titlebar and icon managers are delivered to the application. If the pointer is moved quickly and *twm* is slow to respond, input can be directed to the old window instead of the new. This option is typically used to prevent this ''input lag'' and to work around bugs in older applications that have problems with focus events.

NoTitleHighlight [{ *win-list* }]
This variable indicates that the highlight area of the titlebar, which is used to indicate the window that currently has the input focus, should not be displayed. If the optional *win-list* is given, only those windows will not have highlight areas. This and the **SqueezeTitle** options can be set to substantially reduce the amount of screen space required by titlebars.

OpaqueMove
This variable indicates that the **f.move** function should actually move the window instead of just an outline so that the user can immediately see what the window will look like in the new position. This option is typically used on fast displays (particularly if **NoGrabServer** is set).

Pixmaps { *pixmaps* }
This variable specifies a list of pixmaps that define the appearance of various images. Each entry is a keyword indicating the pixmap to set, followed by a string giving the name of the bitmap file. The following pixmaps may be specified:

```
Pixmaps
{
        TitleHighlight      "gray1"
}
```

The default for *TitleHighlight* is to use an even stipple pattern.

RandomPlacement
This variable indicates that windows with no specified geometry should should be placed in a pseudo-random location instead of having the user drag out an outline.

ResizeFont *string*
This variable specifies the font to be used for in the dimensions window when resizing windows. The default is "fixed".

RestartPreviousState
This variable indicates that *twm* should attempt to use the WM_STATE property on client windows to tell which windows should be iconified and which should be left visible. This is typically used to try to regenerate the state that the screen was in before the previous window manager was shutdown.

SaveColor { *colors-list* }

This variable indicates a list of color assignments to be stored as pixel values in the root window property _MIT_PRIORITY_COLORS. Clients may elect to preserve these values when installing their own colormap. Note that use of this mechanism is a way an for application to avoid the "technicolor" problem, whereby useful screen objects such as window borders and titlebars disappear when a programs custom colors are installed by the window manager. For example:

```
SaveColor
{
        BorderColor
        TitleBackground
        TitleForeground
        "red"
        "green"
        "blue"
}
```

This would place on the root window 3 pixel values for borders and titlebars, as well as the three color strings, all taken from the default colormap.

ShowIconManager

This variable indicates that the icon manager window should be displayed when *twm* is started. It can always be brought up using the **f.showiconmgr** function.

SortIconManager

This variable indicates that entries in the icon manager should be sorted alphabetically rather than by simply appending new windows to the end.

SqueezeTitle [{ *squeeze-list* }]

This variable indicates that *twm* should attempt to use the SHAPE extension to make titlebars occupy only as much screen space as they need, rather than extending all the way across the top of the window. The optional *squeeze-list* may be used to control the location of the squeezed titlebar along the top of the window. It contains entries of the form:

"*name*" *justification* *num* *denom*

where *name* is a window name, *justification* is either **left**, **center**, or **right**, and *num* and *denom* are numbers specifying a ratio giving the relative position about which the titlebar is justified. The ratio is measured from left to right if the numerator is positive, and right to left if negative. A denominator of 0 indicates that the numerator should be measured in pixels. For convenience, the ratio 0/0 is the same as 1/2 for **center** and -1/1 for **right**. For example:

```
SqueezeTitle
{
       "XTerm"       left          0      0
```

```
        "xterm1"      left          1      3
        "xterm2"      left          2      3
        "oclock"      center        0      0
        "emacs"       right         0      0
}
```

The **DontSqueezeTitle** list can be used to turn off squeezing on certain titles.

StartIconified [{ *win-list* }]
This variable indicates that client windows should initially be left as icons until explicitly deiconified by the user. If the optional *win-list* is given, only those windows will be started iconic. This is useful for programs that do not support an *-iconic* command line option or resource.

TitleBackground *string* [{ *win-list* }]
This variable specifies the background color used in titlebars, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. The default is "white".

TitleButtonBorderWidth *pixels*
This variable specifies the width in pixels of the border surrounding titlebuttons. This is typically set to 0 to allow titlebuttons to take up as much space as possible and to not have a border. The default is 1.

TitleFont *string*
This variable specifies the font to used for displaying window names in titlebars. The default is "variable".

TitleForeground *string* [{ *win-list* }]
This variable specifies the foreground color used in titlebars, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. The default is "black".

TitlePadding *pixels*
This variable specifies the distance between the various buttons, text, and highlight areas in the titlebar. The default is 8 pixels.

UnknownIcon *string*
This variable specifies the filename of a bitmap file to be used as the default icon. This bitmap will be used as the icon of all clients which do not provide an icon bitmap and are not listed in the **Icons** list.

UsePPosition *string*
This variable specifies whether or not *twm* should honor program-requested locations (given by the **PPosition** flag in the WM_NORMAL_HINTS property) in the absence of a user-specified position. The argument *string* may have one of three values: "**off**" (the default) indicating that *twm* should ignore the program-supplied position, "**on**" indicating that the position should be used, and "**non-zero**" indicating that the position should used if it is other than (0,0). The latter option is for working around a bug in older toolkits.

WarpCursor [{ *win-list* }]
This variable indicates that the pointer should be warped into windows when they are deiconified. If the optional *win-list* is given, the pointer will only be warped when those windows are deiconified.

WindowRing { *win-list* }
This variable specifies a list of windows along which the **f.warpring** function cycles.

WarpUnmapped
This variable indicates that that the **f.warpto** function should deiconify any iconified windows it encounters. This is typically used to make a key binding that will pop a particular window (such as *xmh*), no matter where it is. The default is for **f.warpto** to ignore iconified windows.

XorValue *number*
This variable specifies the value to use when drawing window outlines for moving and resizing. This should be set to a value that will result in a variety of of distinguishable colors when exclusive-or'ed with the contents of the user's typical screen. Setting this variable to 1 often gives nice results if adjacent colors in the default colormap are distinct. By default, *twm* will attempt to cause temporary lines to appear at the opposite end of the colormap from the graphics.

Zoom [*count*]
This variable indicates that outlines suggesting movement of a window to and from its iconified state should be displayed whenever a window is iconified or deiconified. The optional *count* argument specifies the number of outlines to be drawn. The default count is 8.

The following variables must be set after the fonts have been assigned, so it is usually best to put them at the end of the variables or beginning of the bindings sections:

DefaultFunction *function*
This variable specifies the function to be executed when a key or button event is received for which no binding is provided. This is typically bound to **f.nop**, **f.beep**, or a menu containing window operations.

WindowFunction *function*
This variable specifies the function to execute when a window is selected from the **TwmWindows** menu. If this variable is not set, the window will be deiconified and raised.

BINDINGS

After the desired variables have been set, functions may be attached titlebuttons and key and pointer buttons. Titlebuttons may be added from the left or right side and appear in the titlebar from left-to-right according to the order in which they are specified. Key and pointer button bindings may be given in any order.

Titlebuttons specifications must include the name of the pixmap to use in the button box and the function to be invoked when a pointer button is pressed

within them:

```
LeftTitleButton "bitmapname"      = function
or
RightTitleButton "bitmapname"     = function
```

The *bitmapname* may refer to one of the built-in bitmaps (which are scaled to match **TitleFont**) by using the appropriate colon-prefixed name described above.

Key and pointer button specifications must give the modifiers that must be pressed, over which parts of the screen the pointer must be, and what function is to be invoked. Keys are given as strings containing the appropriate keysym name; buttons are given as the keywords **Button1**-**Button5**:

```
"FP1"         = modlist : context : function
Button1       = modlist : context : function
```

The *modlist* is any combination of the modifier names **shift**, **control**, **lock**, **meta**, **mod1**, **mod2**, **mod3**, **mod4**, or **mod5** (which may be abbreviated as **s**, **c**, **l**, **m**, **m1**, **m2**, **m3**, **m4**, **m5**, respectively) separated by a vertical bar (|). Similarly, the *context* is any combination of **window**, **title**, **icon**, **root**, **frame**, **iconmgr**, their first letters (**iconmgr** abbreviation is **m**), or **all**, separated by a vertical bar. The *function* is any of the **f.** keywords described below. For example, the default startup file contains the following bindings:

```
Button1       =      : root         : f.menu "TwmWindows"
Button1       = m    : window | icon      : f.function "move-or-lower"
Button2       = m    : window | icon      : f.iconify
Button3       = m    : window | icon      : f.function "move-or-raise"
Button1       =      : title              : f.function "move-or-raise"
Button2       =      : title              : f.raiselower
Button1       =      : icon         : f.function "move-or-iconify"
Button2       =      : icon         : f.iconify
Button1       =      : iconmgr      : f.iconify
Button2       =      : iconmgr      : f.iconify
```

A user who wanted to be able to manipulate windows from the keyboard could use the following bindings:

```
"F1"          =      : all          : f.iconify
"F2"          =      : all          : f.raiselower
"F3"          =      : all          : f.warpring "next"
"F4"          =      : all          : f.warpto "xmh"
"F5"          =      : all          : f.warpto "emacs"
"F6"          =      : all          : f.colormap "next"
"F7"          =      : all          : f.colormap "default"
"F20"         =      : all          : f.warptoscreen "next"
"Left"        = m    : all          : f.backiconmgr
"Right"       = m | s       : all          : f.forwiconmgr
"Up"          = m    : all          : f.upiconmgr
"Down" = m | s       : all          : f.downiconmgr
```

Twm provides many more window manipulation primitives than can be conveniently stored in a titlebar, menu, or set of key bindings. Although a small set of defaults are supplied (unless the **NoDefaults** is specified), most

users will want to have their most common operations bound to key and button strokes. To do this, *twm* associates names with each of the primitives and provides *user-defined functions* for building higher level primitives and *menus* for interactively selecting among groups of functions.

User-defined functions contain the name by which they are referenced in calls to **f.function** and a list of other functions to execute. For example:

```
Function "move-or-lower"   { f.move f.deltastop f.lower }
Function "move-or-raise"   { f.move f.deltastop f.raise }
Function "move-or-iconify" { f.move f.deltastop f.iconify }
Function "restore-colormap"      { f.colormap "default" f.lower }
```

The function name must be used in **f.function** exactly as it appears in the function specification.

In the descriptions below, if the function is said to operate on the selected window, but is invoked from a root menu, the cursor will be changed to the **Select** cursor and the next window to receive a button press will be chosen:

! *string* This is an abbreviation for **f.exec** *string*.

f.autoraise
: This function toggles whether or not the selected window is raised whenever entered by the pointer. See the description of the variable **AutoRaise**.

f.backiconmgr
: This function warps the pointer to the previous column in the current icon manager, wrapping back to the previous row if necessary.

f.beep This function sounds the keyboard bell.

f.bottomzoom
: This function is similar to the **f.fullzoom** function, but resizes the window to fill only the bottom half of the screen.

f.circledown
: This function lowers the top-most window that occludes another window.

f.circleup
: This function raises the bottom-most window that is occluded by another window.

f.colormap *string*
: This function rotates the colormaps (obtained from the WM_COLORMAP_WINDOWS property on the window) that *twm* will display when the pointer is in this window. The argument *string* may have one of the following values: "**next**", "**prev**", and "**default**". It should be noted here that in general, the installed colormap is determined by keyboard focus. A pointer driven keyboard focus will install a private colormap upon entry of the window owning the colormap. Using the click to type model, private colormaps will not be installed until the user presses a mouse button on the target window.

f.deiconify
: This function deiconifies the selected window. If the window is not an icon, this function does nothing.

f.delete This function sends the WM_DELETE_WINDOW message to the selected window if the client application has requested it through the

WM_PROTOCOLS window property. The application is supposed to respond to the message by removing the indicated window. If the window has not requested WM_DELETE_WINDOW messages, the keyboard bell will be rung indicating that the user should choose an alternative method. Note this is very different from f.destroy. The intent here is to delete a single window, not necessarily the entire application.

f.deltastop
This function allows a user-defined function to be aborted if the pointer has been moved more than *MoveDelta* pixels. See the example definition given for **Function** "**move-or-raise**" at the beginning of the section.

f.destroy
This function instructs the X server to close the display connection of the client that created the selected window. This should only be used as a last resort for shutting down runaway clients. See also f.delete.

f.downiconmgr
This function warps the pointer to the next row in the current icon manger, wrapping to the beginning of the next column if necessary.

f.exec
This function passes the argument *string* to /bin/sh for execution. In multiscreen mode, if *string* starts a new X client without giving a display argument, the client will appear on the screen from which this function was invoked.

f.focus This function toggles the keyboard focus of the server to the selected window, changing the focus rule from pointer-driven if necessary. If the selected window already was focused, this function executes an **f.unfocus**.

f.forcemove
This function is like **f.move** except that it ignores the **DontMoveOff** variable.

f.forwiconmgr
This function warps the pointer to the next column in the current icon manager, wrapping to the beginning of the next row if necessary.

f.fullzoom
This function resizes the selected window to the full size of the display or else restores the original size if the window was already zoomed.

f.function *string*
This function executes the user-defined function whose name is specified by the argument *string*.

f.hbzoom
This function is a synonym for **f.bottomzoom**.

f.hideiconmgr
This function unmaps the current icon manager.

f.horizoom
This variable is similar to the **f.zoom** function except that the selected window is resized to the full width of the display.

f.htzoom
This function is a synonym for **f.topzoom**.

f.hzoom

This function is a synonym for **f.horizoom**.

f.iconify
This function iconifies or deiconifies the selected window or icon, respectively.

f.identify
This function displays a summary of the name and geometry of the selected window. Clicking the pointer or pressing a key in the window will dismiss it.

f.lefticonmgr
This function similar to **f.backiconmgr** except that wrapping does not change rows.

f.leftzoom
This variable is similar to the **f.bottomzoom** function but causes the selected window is only resized to the left half of the display.

f.lower This function lowers the selected window.

f.menu *string*
This function invokes the menu specified by the argument *string*. Cascaded menus may be built by nesting calls to **f.menu**.

f.move This function drags an outline of the selected window (or the window itself if the **OpaqueMove** variable is set) until the invoking pointer button is released. Double clicking within the number of milliseconds given by **ConstrainedMoveTime** warps the pointer to the center of the window and constrains the move to be either horizontal or vertical depending on which grid line is crossed. To abort a move, press another button before releasing the first button.

f.nexticonmgr
This function warps the pointer to the next icon manager containing any windows on the current or any succeeding screen.

f.nop This function does nothing and is typically used with the **DefaultFunction** or **WindowFunction** variables or to introduce blank lines in menus.

f.previconmgr
This function warps the pointer to the previous icon manager containing any windows on the current or preceding screens.

f.quit This function causes *twm* to restore the window's borders and exit. If *twm* is the first client invoked from *xdm*, this will result in a server reset.

f.raise This function raises the selected window.

f.raiselower
This function raises the selected window to the top of the stacking order if it is occluded by any windows, otherwise the window will be lowered.

f.refresh
This function causes all windows to be refreshed.

f.resize This function displays an outline of the selected window. Crossing a border (or setting **AutoRelativeResize**) will cause the outline to begin to rubber band until the invoking button is released. To abort a resize, press another button before releasing the first button.

f.restart This function kills and restarts *twm*.

f.righticonmgr
This function is similar to **f.nexticonmgr** except that wrapping does

not change rows.

f.rightzoom
This variable is similar to the **f.bottomzoom** function except that the selected window is only resized to the right half of the display.

f.saveyourself
This function sends a WM_SAVEYOURSELF message to the selected window if it has requested the message in its WM_PROTOCOLS window property. Clients that accept this message are supposed to checkpoint all state associated with the window and update the WM_COMMAND property as specified in the ICCCM. If the selected window has not selected for this message, the keyboard bell will be rung.

f.showiconmgr
This function maps the current icon manager.

f.sorticonmgr
This function sorts the entries in the current icon manager alphabetically. See the variable **SortIconManager**. *twm* startup file. *twm* variables are changed.

f.title This function provides a centered, unselectable item in a menu definition. It should not be used in any other context.

f.topzoom
This variable is similar to the **f.bottomzoom** function except that the selected window is only resized to the top half of the display.

f.unfocus
This function resets the focus back to pointer-driven. This should be used when a focused window is no longer desired.

f.upiconmgr
This function warps the pointer to the previous row in the current icon manager, wrapping to the last row in the same column if necessary. *twm* version window to be displayed. This

f.vlzoom
This function is a synonym for **f.leftzoom**.

f.vrzoom
This function is a synonym for **f.rightzoom**.

f.warpring *string*
This function warps the pointer to the next or previous window (as indicated by the argument *string*, which may be "**next**" or "**prev**") specified in the **WindowRing** variable.

f.warpto *string*
This function warps the pointer to the window which has a name or class that matches *string*. If the window is iconified, it will be deiconified if the variable **WarpUnmapped** is set or else ignored.

f.warptoiconmgr *string*
This function warps the pointer to the icon manager entry associated with the window containing the pointer in the icon manager specified by the argument *string*. If *string* is empty (i.e. ""), the current icon manager is chosen.

f.warptoscreen *string*
This function warps the pointer to the screen specified by the argument *string*. *String* may be a number (e.g. "**0**" or "**1**"), the word "**next**" (indicating the current screen plus 1, skipping over any

unmanaged screens), the word "**back**" (indicating the current screen minus 1, skipping over any unmanaged screens), or the word "**prev**" (indicating the last screen visited.

f.winrefresh
This function is similar to the **f.refresh** function except that only the selected window is refreshed.

f.zoom This function is similar to the **f.fullzoom** function, except that the only the height of the selected window is changed.

MENUS

Functions may be grouped and interactively selected using pop-up (when bound to a pointer button) or pull-down (when associated with a titlebutton) menus. Each menu specification contains the name of the menu as it will be referred to by **f.menu**, optional default foreground and background colors, the list of item names and the functions they should invoke, and optional foreground and background colors for individual items:

```
Menu "menuname" [ ("deffore":"defback") ]
{
        string1         [ ("fore1":"backn")]function1
        string6         [ ("fore6":"backn")]function6
                .
        stringN         [ ("foreN":"backN")]functionN
}
```

The *menuname* is case-sensitive. The optional *deffore* and *defback* arguments specify the foreground and background colors used on a color display to highlight menu entries. The *string* portion of each menu entry will be the text which will appear in the menu. The optional *fore* and *back* arguments specify the foreground and background colors of the menu entry when the pointer is not in the entry. These colors will only be used on a color display. The default is to use the colors specified by the **MenuForeground** and **MenuBackground** variables. The *function* portion of the menu entry is one of the functions, including any user-defined functions, or additional menus.

There is a special menu named **TwmWindows** which contains the names of all of the client and *twm*-supplied windows. Selecting an entry will cause the **WindowFunction** to be executed on that window. If **WindowFunction** hasn't been set, the window will be deiconified and raised.

ICONS

Twm supports several different ways of manipulating iconified windows. The common pixmap-and-text style may be laid out by hand or automatically arranged as described by the **IconRegion** variable. In addition, a terse grid of icon names, called an icon manager, provides a more efficient use of screen space as well as the ability to navigate among windows from the keyboard.

An icon manager is a window that contains names of selected or all windows currently on the display. In addition to the window name, a small button using the default iconify symbol will be displayed to the left of the name when the window is iconified. By default, clicking on an entry in the icon manager performs **f.iconify**. To change the actions taken in the icon manager, use the the **iconmgr** context when specifying button and keyboard bindings.

Moving the pointer into the icon manager also directs keyboard focus to the indicated window (setting the focus explicitly or else sending synthetic events **NoTitleFocus** is set). Using the **f.upiconmgr**, **f.downiconmgr f.lefticonmgr**, and **f.righticonmgr** functions, the input focus can be changed between windows directly from the keyboard.

BUGS

The resource manager should have been used instead of all of the window lists.

The **IconRegion** variable should take a list.

Double clicking very fast to get the constrained move function will sometimes cause the window to move, even though the pointer is not moved.

If **IconifyByUnmapping** is on and windows are listed in **IconManagerDontShow** but not in **DontIconifyByUnmapping**, they may be lost if they are iconified and no bindings to **f.menu** "**TwmWindows**" or **f.warpto** are setup.

FILES

$HOME/.twmrc.<*screen_number*>
$HOME/.twmrc
$OPENNT_ROOT/usr/lib/X11/twm/system.twmrc

ENVIRONMENT VARIABLES

DISPLAY
: This variable is used to determine which X server to use. It is also set during **f.exec** so that programs come up on the proper screen.

HOME
: This variable is used as the prefix for files that begin with a tilde and for locating the *twm* startup file.

SEE ALSO

X, *Xserver*, *xdm*, *xrdb*

COPYRIGHT

Portions copyright 1988 Evans & Sutherland Computer Corporation; portions copyright 1989 Hewlett-Packard Company and the Massachusetts Institute of Technology, See *X* for a full statement of rights and permissions.

AUTHORS

Tom LaStrange, Solbourne Computer; Jim Fulton, MIT X Consortium; Steve Pitschke, Stardent Computer; Keith Packard, MIT X Consortium; Dave Sternlicht, MIT X Consortium; Dave Payne, Apple Computer.

NAME

umask – set or report file mode creation mask

SYNOPSIS

`umask [-S] [`*`mask`*`]`

DESCRIPTION

The *umask* command sets or reports the file mode creation mask for the current shell execution environment. The file mode creation mask is used to determine the initial value of a file's permissions when it is created.

The following option is available:

–S Report the file mode creation mask symbolically. If **–S** is not used, then the output format is unspecified.

The output format for **–S** is:

`"u=%s,g=%s,o=%s\n",` *owner perms, group perms, other perms*

The *mask* argument can be specified as either a symbolic mode, or (obsolescent) octal number. Please see the discussion of *modes* in the *chmod* reference page. The file mode creation mask of the shell execution environment will be set according to the *mask* argument. For a *symbolic mode* value, the new value of the file mode creation mask will be the logical complement of the file permissions portion of the file mode specified as a *symbolic mode*.

The output from an invocation of *umask* without a *mask* argument can be used as input to another invocation of *umask*.

If the *mask* argument is not given, then *umask* writes the value of the current file mode creation mask to standard output.

SEE ALSO

chmod

NAME

uname – display information about the system

SYNOPSIS

```
uname [-amnrsv]
```

DESCRIPTION

The *uname* command writes the name of the operating system implementation to standard output. When options are specified, strings representing one or more system characteristics are written to standard output.

The options are as follows:

–a Behave as though the options **–m**, **–n**, **–r**, **–s**, and **–v** were specified. For example, *uname* produces the output:

```
Windows_NT STUD 4.0 SP0 Pentium
```

for a Windows NT machine named Stud, running version 4.0, Service Pack 0 on a Pentium platform.

–m Write the type of the current hardware platform to standard output. The value is one of "i386", "i486", "Pentium", "Pentium II", "Alpha 21064" or "Alpha 21164".

–n Write the name of the system to standard output.

–r Write the current release level of the operating system to standard output. For example, on a Windows NT 4.0 machine, this is "4.0".

–s Write the name of the operating system implementation to standard output. On an INTERIX system, this is "Windows_NT".

–v Write the version level of this release of the operating system to standard output. This is the Service Pack number applied to the Windows NT operating system. If no service pack has been applied, it is SP0.

If the **–a** flag is specified, or multiple flags are specified, all output is written on a single line, separated by spaces.

DIAGNOSTICS

The *uname* utility exits 0 on success, and >0 if an error occurs.

NAME

unexpand – compress spaces into tabs

SYNOPSIS

unexpand **[**–a**|**–t *tablist***]** **[***file ...***]**

DESCRIPTION

Unexpand converts spaces into tabs in the data from the standard input or the named files and writes the result on the standard output. Backspace characters are preserved into the output and decrement the column count for tab calculations. Without options, *unexpand* only converts leading blank characters, those at the beginnings of lines.

Unexpand takes these options:

–a By default, only leading blanks and tabs are reconverted to maximal strings of tabs. If the **–a** option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

–t *tablist*

Specify tab stops according to *tablist*. The *tablist* is one or more positive decimal integers. If there is only one number, tabs are set *tablist* columns apart instead of 8 (the default).

If more than one number is specified, they are taken as the positions of the tabstops. The numbers *must* be in ascending order.

–*tabstop*

If a single *tabstop* argument is given, then tabs are set *tabstop* spaces apart instead of the default 8.

If multiple tabstops are given (separated by blank characters or commas), then the tabs are set at those specific columns. Blank characters beyond the last specified tabstop are not converted.

If **–t** is specified, **–a** is ignored.

DIAGNOSTICS

Unexpand exits with status 0 on success and >0 if an error occurred, such as a problem opening a *file*.

SEE ALSO

expand

NAME

unifdef – remove ifdef'ed lines

SYNOPSIS

```
unifdef [-clt] [-D sym -U sym -iD sym -iU sym] ...
        [file]
```

DESCRIPTION

Unifdef is useful for removing **ifdef**ed lines from a file while otherwise leaving the file alone. *Unifdef* acts on **#ifdef**, **#ifndef**, **#else**, and **#endif** lines, and it knows only enough about C to know when one of these is inactive because it is inside a comment, or a single or double quote. Parsing for quotes is very simplistic: when it finds an open quote, it ignores everything (except escaped quotes) until it finds a close quote, and it will not complain if it gets to the end of a line and finds no backslash for continuation.

Available options:

–c If the **–c** flag is specified, then the operation of *unifdef* is complemented, i.e. the lines that would have been removed or blanked are retained and vice versa.

–D*sym*

–U*sym* Specify which symbols to define or undefine. The lines inside those ifdefs will be copied to the output or removed as appropriate. The ifdef, ifndef, else, and endif lines associated with *sym* will also be removed. Ifdefs involving symbols you don't specify and ''#if'' control lines are untouched and copied out along with their associated ifdef, else, and endif lines. If an ifdef X occurs nested inside another ifdef X, then the inside ifdef is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, the last occurrence dominates.

–iD*sym*

–iU*sym* Ignore the **ifdef**s associated with *sym*. If your C code uses **ifdef**s to delimit non-C lines, such as comments or code which is under construction, then you must tell *unifdef* which symbols are used for that purpose so that it won't try to parse for quotes and comments inside those **ifdef**s. Specify ignored **ifdef**s with **–iD** *sym* and **–iU** *sym*, similar to **–D** *sym* and **–U** *sym* above.

–l Replace removed lines with blank lines instead of deleting them.

–t Disables parsing for C comments and quotes, which is useful for plain text.

Unifdef copies its output to stdout and will take its input from stdin if no *file* argument is given.

Unifdef works nicely with the **–D** *sym* option added to *diff* as of the 4.1 Berkeley Software Distribution.

DIAGNOSTICS

Inappropriate else or endif.

Premature EOF with line numbers of the unterminated #ifdefs.

Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

BUGS

Should try to deal with ‘‘#if’’ lines.

Doesn’t work correctly if input contains null characters.

SEE ALSO

diff

NAME

uniq – report or filter out repeated lines in a file

SYNOPSIS

```
uniq [-c|-d|-u] [-f fields] [-s chars]
     [input_file[output_file]]
```

DESCRIPTION

The *uniq* utility reads the standard input comparing adjacent lines, and writes a copy of each unique input line to the standard output. The second and succeeding copies of identical adjacent input lines are not written. Repeated lines in the input will not be detected if they are not adjacent, so it may be necessary to sort the files first.

The following options are available:

–c Precede each output line with the count of the number of times the line occurred in the input, followed by a single space.

–d Don't output lines that are not repeated in the input.

–f *fields* Ignore the first *fields* in each input line when doing comparisons. A field is a string of non-blank characters separated from adjacent fields by blanks. Field numbers are one based, i.e. the first field is field one.

–s *chars* Ignore the first *chars* characters in each input line when doing comparisons. If specified in conjunction with the **–f** option, the first *chars* characters after the first *fields* fields will be ignored. Character numbers are one based, i.e. the first character is character one.

–u Don't output lines that are repeated in the input.

If additional arguments are specified on the command line, the first such argument is used as the name of an input file, the second is used as the name of an output file.

DIAGNOSTICS

The *uniq* utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

sort

NAME

unpack – expand files compressed with pack

SYNOPSIS

unpack *file* ...

DESCRIPTION

The *unpack* utility uncompresses files compressed with *pack*. It replaces the compressed files with their unpacked form. The specified *file* may or may not have the **.z** extension.

The file may not be unpacked for one of the following reasons:

- It cannot be opened.
- It does not appear to have been compressed by *pack*.
- A file with the unpacked name already exists.
- The filename without the **.z** extension is more than {NAME_MAX} bytes long. This is possible if the packed file was brought from another system.

DIAGNOSTICS

The *unpack* utility exits with status 0 on success, and >0 if an error occurred.

NOTES

This utility is actually a link to the *gunzip* command (a copy of the binary on FAT systems).

SEE ALSO

pack, *pcat*, *uncompress*

NAME

unzip – list, test and extract compressed files in a ZIP archive

SYNOPSIS

```
unzip [–Z] [–cflptuvz[ajnoqsUV$]] file[.zip]
      [file(s) ...] [–x xfile(s) ...] [–d exdir]
```

DESCRIPTION

Unzip will list, test, or extract files from a ZIP archive, commonly found on MS-DOS systems. The default behavior (with no options) is to extract into the current directory (and subdirectories below it) all files from the specified ZIP archive. A companion program, *zip*, creates ZIP archives; both programs are compatible with PKWARE's PKZIP and PKUNZIP for MS-DOS.

The main operands are the archive *file*, the members, a list of members to be excluded from processing, and a directory in which the extraction should take place. Only the archive is required.

file[**.zip**]

The path of the ZIP archive(s). If the file specification is a wildcard, each matching file is processed in an order determined by the operating system (or file system). Only the filename can be a wildcard; the path itself cannot. Wildcard expressions are similar to *egrep* (regular) expressions and may contain:

*	matches a sequence of 0 or more characters
?	matches exactly 1 character
[...]	matches any single character found inside the brackets; ranges are specified by a beginning character, a hyphen, and an ending character. If an exclamation point or a caret ('!' or '^') follows the left bracket, then the range of characters within the brackets is complemented (that is, anything *except* the characters inside the brackets is considered a match).

(Be sure to quote any character which might otherwise be interpreted or modified by the operating system or shells.) If no matches are found, the specification is assumed to be a literal filename; and if that also fails, the suffix **.zip** is appended. Note that self-extracting ZIP files are supported; just specify the **.exe** suffix (if any) explicitly.

*file*s The *file*s, if specified, are the archive members to be processed. You can match multiple members with regular expressions (wildcards) as above. Again, be sure to quote expressions that would otherwise be expanded

–d *exdir*

Extract files in the directory *exdir*. By default, all files and subdirectories are recreated in the current directory; the **–d** option allows extraction in an arbitrary directory (always assuming one has permission to write to the directory). This option need not appear at the end of the command line; it is also accepted immediately after the zipfile specification, or between the *file*(s) and the **–x** option. or modified by the shell.

–x *xfile(s)*

Exclude the archive members *xfile(s)* from processing. Since wildcard characters match directory separators (‘/’), this option may be used to exclude any files which are in subdirectories. For example, ‘‘unzip foo *.[ch] -x */*’’ would extract all C source files in the main directory, but none in any subdirectories. Without the **–x** option, all C source files in all directories within the zipfile would be extracted.

Options

Besides those operands, *unzip* also accepts the following options which specify actions:

–c Extract files to stdout/screen (‘‘CRT’’). This option is similar to the **–p** option except that the name of each file is printed as it is extracted, the **–a** option is allowed, and ASCII-EBCDIC conversion is automatically performed if appropriate.

–f Freshen existing files, i.e., extract only those files which already exist on disk and which are newer than the disk copies. By default *unzip* queries before overwriting, but the **–o** option may be used to suppress the queries.

–l List archive files (short format). The name, uncompressed file size and modification date and time of each specified file is printed, along with totals for all files specified. If a file was archived from a single-case file system (for example, the MS-DOS FAT file system) and the **–U** option was not given, the filename is converted to lowercase and is prefixed with a caret (^). In addition, the archive comment and individual file comments (if any) are displayed.

–p Extract files to pipe (stdout). Nothing but the file data is sent to stdout, and the files are always extracted in binary format, just as they are stored (no conversions).

–t Test archive files. This option extracts each specified file in memory and compares the CRC (cyclic redundancy check, an enhanced checksum) of the expanded file with the original file’s stored CRC value.

–u Update existing files and create new ones if needed. This option performs the same function as the **–f** option, extracting (with query) files which are newer than those with the same name on disk, and in addition it extracts those files which do not already exist on disk.

–v List archive files (verbose format). In addition to the information given by the **–l** option, the compression method, compressed size, compression ratio and 32-bit CRC is listed.

–z Display only the archive comment.

–Z *zipinfo* mode. If the first option on the command line is **–Z**, the remaining options are taken to be *zipinfo* options. See the appropriate manual page for a description of these options.

Modifiers

The following options modify the behavior of one or more actions:

–a Convert text files. Ordinarily all files are extracted exactly as they are stored (as ''binary'' files). The **–a** option causes files identified by *zip* as text files (those with the 't' label in *zipinfo* listings, rather than 'b') to be automatically extracted as such, converting line endings, end-of-file characters and the character set itself as necessary. (For example, Interix files use line feeds (LFs) for end-of-line (EOL) and have no end-of-file (EOF) marker; Macintoshes use carriage returns (CRs) for EOLs; and most PC operating systems use CR+LF for EOLs and control-Z for EOF. In addition, IBM mainframes and the Michigan Terminal System use EBCDIC rather than the more common ASCII character set, and NT supports Unicode.) Note that *zip*'s identification of text files is by no means perfect; some ''text'' files may actually be binary and vice versa. *unzip* therefore prints ''[text]'' or ''[binary]'' as a visual check for each file it extracts when using the **–a** option. The **–aa** option forces all files to be extracted as text, regardless of the supposed file type.

–j Junk paths. The archive's directory structure is not recreated; all files are deposited in the extraction directory (by default, the current one).

–n Never overwrite existing files. If a file already exists, skip the extraction of that file without prompting. By default *unzip* queries before extracting any file which already exists; the user may choose to overwrite only the current file, overwrite all files, skip extraction of the current file, skip extraction of all existing files, or rename the current file.

–o Overwrite existing files without prompting. This is a dangerous option, so use it with care. (It is often used with **–f**, however.)

–q Perform operations quietly (**–qq**=even quieter). Ordinarily *unzip* prints the names of the files it's extracting or testing, the extraction methods, any file or zipfile comments which may be stored in the archive, and possibly a summary when finished with each archive. The **–q** and **–qq** options suppress the printing of some or all of these messages.

–s [OS/2, NT, MS-DOS] Convert spaces in filenames to underscores. Since all PC operating systems allow spaces in filenames, *unzip* by default extracts filenames with spaces intact (e.g., ''EA DATA. SF''). This can be awkward, however, since MS-DOS in particular does not gracefully support spaces in filenames. Conversion of spaces to underscores can eliminate the awkwardness in some cases.

–U Leave filenames uppercase if created under MS-DOS, VMS, etc. Depending on the archiver, files archived under single-case filesystems may be stored as all-uppercase names; this can be awkward when extracting to a case-preserving filesystem such as OS/2 HPFS or a case-sensitive one such as under Interix. By default *unzip* converts such filenames to lowercase; this option causes all filenames to be extracted exactly as they're stored (excepting

truncation, conversion of unsupported characters, etc.).

–V Retain (VMS) file version numbers. VMS files can be stored with a version number, in the format *file.ext;##* . By default the ‘‘;##’’ version numbers are stripped, but this option allows them to be retained. (On filesystems which limit filenames to particularly short lengths, the version numbers may be truncated or stripped regardless of this option.)

–X [VMS] Restore owner / protection info (may require system privileges). Ordinary file attributes are always restored, but this option allows UICs to be restored as well. [The next version of *unzip* will support the UID / GID info used with traditional open systems as well, and possibly NT permissions.]

–$** [MS-DOS, OS / 2, NT, Amiga] Restore the volume label if the extraction medium is removable (e.g., a diskette). Doubling the option (**–$$) allows fixed media (hard disks) to be labelled as well. By default, volume labels are ignored.

ENVIRONMENT OPTIONS

Unzip’s default behavior may be modified via options placed in an environment variable. This can be done with any option, but it is probably most useful with the **–q**, **–a**, **–o**, or **–n** modifiers: make *unzip* quieter by default, make it auto-convert text files, or make it always overwrite or never overwrite files as it extracts them. For example, to make *unzip* act as quietly as possible, only reporting errors, one would use one of the following commands:

```
UNZIP=–qq; export UNZIP     Bourne shell
setenv UNZIP –qq            BSD C shell
```

Environment options are, in effect, considered to be just like any other command-line options, except that they are effectively the first options on the command line. To override an environment option, one may use the ‘‘minus operator’’ to remove it. For instance, to override one of the quiet-flags in the example above, use the command

```
unzip ––q[other options] zipfile
```

The first hyphen is the normal switch character, and the second is a minus sign, acting on the q option. Thus the effect here is to cancel one quantum of quietness. To cancel both quiet flags, two (or more) minuses may be used:

```
unzip –t––q zipfile
unzip –––qt zipfile
```

(the two are equivalent). This may seem awkward or confusing, but it is reasonably intuitive: just ignore the first hyphen and go from there.

EXAMPLES

To use *unzip* to extract all members of the archive *letters.zip* into the current directory and subdirectories below it, creating any subdirectories as necessary:

```
unzip letters"
```

To extract all members of **letters.zip** into the current directory only:

```
unzip -j letters
```

To test **letters.zip**, printing only a summary message indicating whether the archive is OK or not:

```
unzip -tq letters
```

To extract to standard output all members of *letters.zip* whose names end in *.tex*, auto-converting to the local end-of-line convention and piping the output into *more*(1):

```
unzip –ca letters \*.tex | more"
```

(The backslash before the asterisk is required when the shell expands wildcards, as with Interix; double quotes could have been used instead, as in the source example below.) To extract the binary file *paper1.dvi* to standard output and pipe it to a printing program:

```
unzip –p articles paper1.dvi | dvips
```

To extract all FORTRAN and C source files--*.f, *.c, *.h, and Makefile--into the /tmp directory:

```
unzip source.zip "*.[fch]"" Makefile -d /tmp"
```

(the double quotes are necessary to have the shell treat the enclosed characters as literals). To extract only newer versions of the files already in the current directory, without querying (NOTE: be careful of unzipping in one timezone a zipfile created in another--ZIP archives to date contain no timezone information, and a ‘‘newer’’ file from an eastern timezone may, in fact, be older):

```
unzip –fo sources
```

To extract newer versions of the files already in the current directory and to create any files not already there (same caveat as previous example):

```
unzip –uo sources"
```

In the last five examples, assume that UNZIP or UNZIP_OPTS is set to -q. To do a singly quiet listing:

```
unzip -l file
```

To do a doubly quiet listing:

```
unzip -ql file
```

To do a standard listing:

```
unzip --ql file
```

or

```
unzip -l-q file
```

or

```
unzip -l--q file      (extra minuses don’t hurt)
```

TIPS

The current maintainer, being a lazy sort, finds it very useful to define a pair of aliases: *tt* for ‘‘unzip –tq’’ and *ii* for ‘‘unzip –Z’’ (or ‘‘zipinfo’’). One may then

simply type ‘‘tt zipfile’’ to test an archive, something which is worth making a habit of doing. With luck *unzip* will report ‘‘No errors detected in zipfile.zip,’’ after which one may breathe a sigh of relief.

BUGS

Under DEC Ultrix, *unzip* will sometimes fail on long zipfiles (bad CRC, not always reproducible). This is apparently due either to a hardware bug (cache memory) or an operating system bug (improper handling of page faults?).

Dates and times of stored directories are not restored.

SEE ALSO

funzip, *zip*, *zipcloak*, *zipgrep*, *zipinfo*, *zipnote*, *zipsplit*

AUTHORS

The primary Info-ZIP authors (current zip-bugs workgroup) are: Jean-loup Gailly (Zip), Greg R. Roelofs (UnZip), Mark Adler (decompression, fUnZip), Kai Uwe Rommel (OS/2), Igor Mandrichenko (VMS), John Bush and Paul Kienitz (Amiga), Antoine Verheijen (Macintosh), Chris Herborth (Atari), Henry Gessau (NT), and Robert Heath (Windows). The author of the original unzip code upon which Info-ZIP’s is based was Samuel H. Smith; Carl Mascott did the first open systems port; and David P. Kirschbaum organized and led Info-ZIP in its early days. The full list of contributors to UnZip has grown quite large; please refer to the CONTRIBS file in the UnZip source distribution for a more complete list.

VERSIONS

v1.215 Mar 89	Samuel H. Smith
v2.0 9 Sep 89	Samuel H. Smith
v2.xfall 1989	many Usenet contributors
v3.0 1 May 90	Info-ZIP (DPK, consolidator)
v3.115 Aug 90	Info-ZIP (DPK, consolidator)
v4.0 1 Dec 90	Info-ZIP (GRR, maintainer)
v4.112 May 91	Info-ZIP
v4.220 Mar 92	Info-ZIP (zip-bugs subgroup; GRR, maint.)
v5.021 Aug 92	Info-ZIP (zip-bugs subgroup; GRR, maint.)
v5.1 7 Feb 94	Info-ZIP (zip-bugs subgroup; GRR, maint.)

NAME

uuencode, uudecode – encode / decode a binary file

SYNOPSIS

```
uuencode [file] name
uudecode [file ...]
```

DESCRIPTION

Uuencode and *uudecode* are used to transmit binary files over transmission mediums that do not support other than simple ASCII data.

Uuencode reads *file* (or by default the standard input) and writes an encoded version to the standard output. The *name* operand specifies the name of the file when decoded (by *uudecode*). The encoding uses only printing ASCII characters and includes the mode of the file and the operand *name* for use by *uudecode*.

Uudecode transforms *uuencoded* files (or by default, the standard input) into the original form. The resulting file is named *name* and will have the mode of the original file except that setuid and execute bits are not retained. *Uudecode* ignores any leading and trailing lines.

Uuencode and *uudecode* are often used for mailing binaries. In *mailx*'s mail-reading mode, you can decode the current message with the command

```
| uudecode
```

EXAMPLE

To encode the file **samples.zip** and store in **sample.zip.uue**:

```
$ uuencode samples.zip samples.zip > samples.zip.uue
```

To encode the output of *xwd* and mail to user chris:

```
$ xwd | uuencode screenshot.xwd | mailx chris
```

To decode the file **samples.zip.uue**:

```
$ uudecode samples.zip.uue
```

SEE ALSO

gzip

NAME

ex, vi, view – text editors

SYNOPSIS

`ex [-eFRrsv] [-c cmd] [-t tag] [-w size] [file ...]`

`vi [-eFlRrv] [-c cmd] [-t tag] [-w size] [file ...]`

`view [-eFRrv] [-c cmd] [-t tag] [-w size] [file ...]`

DESCRIPTION

Vi is a screen oriented text editor. *Ex* is a line-oriented text editor. *Ex* and *vi* are different interfaces to the same program, and it is possible to switch back and forth during an edit session. *View* is the equivalent of using the **–R** (read-only) option of *vi*.

This manual page is the one provided with the *nex* / *nvi* versions of the *ex* / *vi* text editors. *Nex* / *nvi* are intended as bug-for-bug compatible replacements for the original Fourth Berkeley Software Distribution (4BSD) *ex* and *vi* programs. For the rest of this manual page, *nex* / *nvi* is used only when it's necessary to distinguish it from the historic implementations of *ex/vi*.

This manual page is intended for users already familiar with *ex* / *vi*. Anyone else should almost certainly read a good tutorial on the editor before this manual page. If you're in an unfamiliar environment, and you absolutely have to get work done immediately, read the section after the options description, entitled "Fast Startup". It's probably enough to get you going.

The following options are available:

+*cmd*

–c *cmd* Execute *cmd* immediately after starting the edit session. Particularly useful for initial positioning in the file, however *cmd* is not limited to positioning commands. This is the POSIX 1003.2 interface for the historic **+***cmd* syntax. *Nex* / *nvi* supports both the old and new syntax.

–e Start editing in ex mode, as if the command name were *ex*.

–F Don't copy the entire file when first starting to edit. (The default is to make a copy in case someone else modifies the file during your edit session.)

–l Start editing with the lisp and showmatch options set.

–R Start editing in read-only mode, as if the command name was *view*, or the **readonly** option was set.

–r Recover the specified files, or, if no files are specified, list the files that could be recovered. If no recoverable files by the specified name exist, the file is edited as if the **–r** option had not been specified.

–s Enter batch mode; applicable only to *ex* edit sessions. Batch mode is useful when running *ex* scripts. Prompts, informative messages and other user oriented message are turned off, and no startup files or environmental variables are read. This is the POSIX 1003.2 interface for the historic "–" argument. *Nex* / *nvi* supports both the old and new

syntax.

–t Start editing at the specified tag. (See *ctags*).

–w Set the initial window size to the specified number of lines.

–v Start editing in vi mode, as if the command name was *vi* or *view*.

Command input for *ex* / *vi* is read from the standard input. In the *vi* interface, it is an error if standard input is not a terminal. In the *ex* interface, if standard input is not a terminal, *ex* will read commands from it regardless, however, the session will be a batch mode session, exactly as if the **–s** option had been specified.

FAST STARTUP

This section will tell you the minimum amount that you need to do simple editing tasks using *vi*. If you've never used any screen editor before, you're likely to have problems even with this simple introduction. In that case you should find someone that already knows *vi* and have them walk you through this section.

Vi is a screen editor. This means that it takes up almost the entire screen, displaying part of the file on each screen line, except for the last line of the screen. The last line of the screen is used for you to give commands to *vi*, and for *vi* to give information to you.

The other fact that you need to understand is that *vi* is a modeful editor, i.e. you are either entering text or you are executing commands, and you have to be in the right mode to do one or the other. You will be in command mode when you first start editing a file. There are commands that switch you into input mode. There is only one key that takes you out of input mode, and that is the <escape> key. (Key names are written using less-than and greater-than signs, e.g. <escape> means the "escape" key, usually labeled "esc" on your terminal's keyboard.) If you're ever confused as to which mode you're in, keep entering the <escape> key until *vi* beeps at you. (Generally, *vi* will beep at you if you try and do something that's not allowed. It will also display error messages.)

To start editing a file, enter the command "vi file_name<carriage-return>". The command you should enter as soon as you start editing is ":set verbose showmode<carriage-return>". This will make the editor give you verbose error messages and display the current mode at the bottom of the screen.

The commands to move around the file are:

h Move the cursor left one character.
j Move the cursor down one line.
k Move the cursor up one line.
l Move the cursor right one character.
<cursor-arrows>
The cursor arrow keys should work, too.
/text<carriage-return>
Search for the string "text" in the file, and move the cursor to its first character.

The commands to enter new text are:

a Append new text, *after* the cursor.

i Insert new text, *before* the cursor.
o Open a new line below the line the cursor is on, and start entering text.
O Open a new line above the line the cursor is on, and start entering text.
<escape>
Once you've entered input mode using the one of the **a**, **i**, **O** or **o** commands, use **<escape>** to quit entering text and return to command mode.

The commands to copy text are:

yy Copy the line the cursor is on.
p Append the copied line after the line the cursor is on.

The commands to delete text are:

dd Delete the line the cursor is on.
x Delete the character the cursor is on.

The commands to write the file are:

:w<carriage-return>
Write the file back to the file with the name that you originally used as an argument on the *vi* command line.
:w *file_name***<carriage-return>**
Write the file back to the file with the name *file_name*.

The commands to quit editing and exit the editor are:

:q<carriage-return>
Quit editing and leave *vi* (if you've modified the file, but not saved your changes, *vi* will refuse to quit).
:q!<carriage-return>
Quit, discarding any modifications that you may have made.

One final caution. Unusual characters can take up more than one column on the screen, and long lines can take up more than a single screen line. The above commands work on "physical" characters and lines, i.e. they affect the entire line no matter how many screen lines it takes up and the entire character no matter how many screen columns it takes up.

VI COMMANDS

The following section describes the commands available in the command mode of the *vi* editor. In each entry below, the tag line is a usage synopsis for the command character.

[*count*] **<control-A>**
Search forward *count* times for the current word.
[*count*] **<control-B>**
Page backwards *count* screens.
[*count*] **<control-D>**
Scroll forward *count* lines.
[*count*] **<control-E>**
Scroll forward *count* lines, leaving the current line and column as is, if possible.
[*count*] **<control-F>**
Page forward *count* screens.
<control-G>

Display the file information.

<control-H>

[*count*] **h** Move the cursor back *count* characters in the current line.

[*count*] **<control-J>**

[*count*] **<control-N>**

[*count*] **j** Move the cursor down *count* lines without changing the current column.

<control-L>

<control-R> Repaint the screen.

[*count*] **<control-M>**

[*count*] **+** Move the cursor down *count* lines to the first nonblank character of that line.

[*count*] **<control-P>**

[*count*] **k** Move the cursor up *count* lines, without changing the current column.

<control-T> Return to the most recent tag context.

<control-U> Scroll backwards *count* lines.

<control-W>

Switch to the next lower screen in the window, or, to the first screen if there are no lower screens in the window.

<control-Y> Scroll backwards *count* lines, leaving the current line and column as is, if possible.

<control-Z> Suspend the current editor session.

<escape> Execute *ex* commands or cancel partial commands.

<control-]> Push a tag reference onto the tag stack.

<control-^> Switch to the most recently edited file.

[*count*] **<space>**

[*count*] **l** Move the cursor forward *count* characters without changing the current line.

[*count*] **!** *motion shell-argument(s)*

Replace text with results from a shell command.

[*count*] **# #** | **+** | **-**

Increment or decrement the cursor number.

[*count*] **$** Move the cursor to the end of a line.

% Move to the matching character.

& Repeat the previous substitution command on the current line.

'<*character*>

`<*character*> Return to a context marked by the character <*character*>.

[*count*] **(** Back up *count* sentences.

[*count*] **)** Move forward *count* sentences.

[*count*] **,** Reverse find character *count* times.

[*count*] **-** Move to first nonblank of the previous line, *count* times.

[*count*] **.** Repeat the last *vi* command that modified text.

/RE<carriage-return>

/RE/ [*offset*]**<carriage-return>**

?RE<carriage-return>

?RE? [*offset*]**<carriage-return>**

N

n Search forward or backward for a regular expression.

0 Move to the first character in the current line.

: Execute an ex command.

[*count*] **;** Repeat the last character find *count* times.
[*count*] **<** *motion*
[*count*] **>** *motion*
Shift lines left or right.
@ *buffer* Execute a named buffer.
[*count*] **A** Enter input mode, appending the text after the end of the line.
[*count*] **B** Move backwards *count* bigwords.
[*buffer*] [*count*] **C**
Change text from the current position to the end-of-line.
[*buffer*] **D** Delete text from the current position to the end-of-line.
[*count*] **E** Move forward *count* end-of-bigwords.
[*count*] **F** <*character*>
Search *count* times backward through the current line for <*character*>.
[*count*] **G** Move to line *count*, or the last line of the file if *count* not specified.
[*count*] **H** Move to the screen line *count - 1* lines below the top of the screen.
[*count*] **I** Enter input mode, inserting the text at the beginning of the line.
[*count*] **J** Join lines.
[*count*] **L** Move to the screen line *count - 1* lines above the bottom of the screen.
M Move to the screen line in the middle of the screen.
[*count*] **O** Enter input mode, appending text in a new line above the current line.
[*buffer*] **P** Insert text from a buffer.
Q Exit *vi* (or visual) mode and switch to *ex* mode.
[*count*] **R** Enter input mode, replacing the characters in the current line.
[*buffer*] [*count*] *S*
Substitute *count* lines.
[*count*] **T** <*character*>
Search backwards, *count* times, through the current line for the character *after* the specified <*character*>.
U Restore the current line to its state before the cursor last moved to it.
[*count*] **W** Move forward *count* bigwords.
[*buffer*] [*count*] **X**
Delete *count* characters before the cursor.
[*buffer*] [*count*] **Y**
Copy (or “yank”) *count* lines into the specified buffer.
ZZ Write the file and exit *vi*.
[*count*] **[[** Back up *count* section boundaries.
[*count*] **]]** Move forward *count* section boundaries.
^ Move to first nonblank character on the current line.
[*count*] **_** Move down *count - 1* lines, to the first nonblank character.
[*count*] **a** Enter input mode, appending the text after the cursor.
[*count*] **b** Move backwards *count* words.
[*buffer*] [*count*] **c** *motion*
Change a region of text.
[*buffer*] [*count*] **d** *motion*
Delete a region of text.
[*count*] **e** Move forward *count* end-of-words.
[*count*] **f**<*character*>

Search forward, *count* times, through the rest of the current line for <*character*>.

[*count*] **i** Enter input mode, inserting the text before the cursor.

m <*character*>
Save the current context (line and column) as <*character*>.

[*count*] **o** Enter input mode, appending text in a new line under the current line.

[*buffer*] **p** Append text from a buffer.

[*count*] **r** <*character*>
Replace *count* characters.

[*buffer*] [*count*] **s**
Substitute *count* characters in the current line starting with the current character.

[*count*] **t** <*character*>
Search forward, *count* times, through the current line for the character immediately *before* <*character*>.

u Undo the last change made to the file.

[*count*] **w** Move forward *count* words.

[*buffer*] [*count*] **x**
Delete *count* characters.

[*buffer*] [*count*] **y** *motion*
Copy (or "yank") a text region specified by the *count* and motion into a buffer.

[*count1*] **z** [*count2*] **-** | **.** | **+** | **^** | **<carriage-return>**
Redraw, optionally repositioning and resizing the screen.

[*count*] **{** Move backward *count* paragraphs.

[*count*] **|** Move to a specific *column* position on the current line.

[*count*] **}** Move forward *count* paragraphs.

[*count*] **~** Reverse the case of the next *count* character(s).

[*count*] **~** *motion*
Reverse the case of the characters in a text region specified by the *count* and *motion*.

<interrupt> Interrupt the current operation.

VI TEXT INPUT COMMANDS

The following section describes the commands available in the text input mode of the *vi* editor.

<nul> Replay the previous input.

<control-D>
Erase to the previous **shiftwidth** column boundary.

^<control-D>
Erase all of the autoindent characters, and reset the autoindent level.

0<control-D>
Erase all of the autoindent characters.

<control-T> Insert sufficient <*tab*> and <*space*> characters to move forward to the next **shiftwidth** column boundary.

<erase>
<control-H>
Erase the last character.

<literal next>

Quote the next character.

<escape> Resolve all text input into the file, and return to command mode.

<line erase>

Erase the current line.

<control-W>

<word erase>

Erase the last word. The definition of word is dependent on the **altwerase** and **ttywerase** options.

<control-X>[**0-9A-Fa-f**]+

Insert a character with the specified hexadecimal value into the text.

<interrupt> Interrupt text input mode, returning to command mode.

EX COMMANDS

The following section describes the commands available in the *ex* editor. In each entry below, the tag line is a usage synopsis for the command.

<end-of-file>

Scroll the screen.

! *argument(s)*

[*range*]**!** *argument(s)*

Execute a shell command, or filter lines through a shell command.

" A comment.

[*range*] **nu**[**mber**] [*count*] [*flags*]

[*range*] **#** [*count*] [*flags*]

Display the selected lines, each preceded with its line number.

@ *buffer*

***** *buffer* Execute a buffer.

[*line*] **a**[**ppend**][**!**]

The input text is appended after the specified line.

[*range*] **c**[**hange**][**!**] [*count*]

The input text replaces the specified range.

cs[**cope**] **add** | **find** | **help** | **kill** | **reset**

Execute a Cscope command.

[*range*] **d**[**elete**] [*buffer*] [*count*] [*flags*]

Delete the lines from the file.

di[**splay**] **b**[**uffers**] | c**[**onnections**] | s**[**creens**] | **t**[**ags**]

Display buffers, Cscope connections, screens or tags.

[**Ee**][**dit**][**!**] [+cmd] [*file*]

[**Ee**]**x**[**!**] [**+***cmd***] [***file*]

Edit a different file.

exu[**sage**] [*command*]

Display usage for an *ex* command.

f[**ile**] [*file*] Display and optionally change the file name.

[**Ff**]**g** [*name*] *Vi* mode only. Foreground the specified screen.

[*range*] **g**[**lobal**] */pattern/* [*commands*]

[*range*] **v** */pattern/* [*commands*]

Apply commands to lines matching (or not matching) a pattern.

he[lp] Display a help message.

[*line*] **i**[**nsert**][**!**]

The input text is inserted before the specified line.

[*range*] **j**[**oin**][**!**] [*count*] [*flags*]

Join lines of text together.

[*range*] **l**[**ist**] **[***count***] [***flags*]
Display the lines unambiguously.

map[**!**] [*lhs rhs*]
Define or display maps (for *vi* only).

[*line*] **ma**[**rk**] <*character*>
[*line*] **k** <*character*>
Mark the line with the mark <*character*>.

[*range*] **m**[**ove**] *line*
Move the specified lines after the target *line*.

mk[**exrc**][**!**] *file*
Write the abbreviations, editor options and maps to the specified *file*.

[**Nn**][**ext**][**!**] [*file ...*]
Edit the next file from the argument list.

[*line*] **o**[**pen**] */pattern/* [*flags*]
Enter open mode.

pre[**serve**] Save the file in a form that can later be recovered using the *ex* **–r** option.

[**Pp**]**rev**[**ious**][**!**]
Edit the previous file from the argument list.

[*range*] **p**[**rint**] [*count*] [*flags*]
Display the specified lines.

[*line*] **pu**[**t**] [*buffer*]
Append buffer contents to the current line.

q[**uit**][**!**] End the editing session.

[*line*] **r**[**ead**][**!**] [*file*]
Read a file.

rec[**over**] **file**
Recover *file* if it was previously saved.

res[**ize**] [**+** | **-**]*size*
Vi mode only. Grow or shrink the current screen.

rew[**ind**][**!**] Rewind the argument list.

se[**t**] [*option*[**=**[*value*]] ...] [**no***option* ...] [*option?* ...] [**all**]
Display or set editor options.

sh[**ell**] Run a shell program.

so[**urce**] *file* Read and execute *ex* commands from a file.

[*range*] **s**[**ubstitute**] [*/pattern/replace/*] [*options*] [*count*] [*flags*]
[*range*] **&** [*options*] [*count*] [*flags*]
[*range*] **~** [*options*] [*count*] [*flags*]
Make substitutions.

su[**spend**][**!**]
st[**op**][**!**]
<suspend> Suspend the edit session.

[**Tt**]**a**[**g**][**!**] *tagstring*
Edit the file containing the specified tag.

tagn[**ext**][**!**] Edit the file containing the next context for the current tag.

tagp[**op**][**!**] [*file* | *number*]
Pop to the specified tag in the tags stack.

tagp[**rev**][**!**] Edit the file containing the previous context for the current tag.

unm[**ap**][**!**] *lhs*

Unmap a mapped string.

ve[**rsion**] Display the version of the *ex* / *vi* editor.

[*line*] **vi**[**sual**] [*type*] [*count*] [*flags*]
Ex mode only. Enter *vi*.

Vi[**sual**][**!**] [**+***cmd*] [*file*]
Vi mode only. Edit a new file. *Vi* opens a new edit session in half of the screen window (the half without the cursor). Executing the **Vi** command again subdivides this smaller window. To return to the "parent" window / file, quit the *vi* session (with **q** or **Q** or **ZZ**). This is an extension to traditional *vi*.

viu[**sage**] [*command*]
Display usage for a *vi* command.

[*range*] **w**[**rite**][**!**] [**>>**] [*file*]
[*range*] **w**[**rite**] [**!**] [*file*]
[*range*] **wn**[**!**] [**>>**] [*file*]
[*range*] **wq**[**!**] [**>>**] [*file*]
Write the file. The modifier **!** forces the write even if the file is newer than the buffer. The modifier **>>** appends the buffer to the named *file*. The **wn** command writes and then moves to the next file in the list. The **wq** command writes and then quits.

[*range*] **x**[**it**][**!**] [*file*]
Write the file if it has been modified.

[*range*] **ya**[**nk**] [*buffer*] [*count*]
Copy the specified lines to a buffer.

[*line*] **z** [*type*] [*count*] [*flags*]
Adjust the window.

SET OPTIONS

There are a large number of options that may be set (or unset) to change the editor's behavior. This section describes the options, their abbreviations and their default values.

In each entry below, the first part of the tag line is the full name of the option, followed by any equivalent abbreviations. The part in square brackets is the default value of the option. Most of the options are boolean, i.e. they are either on or off, and do not have an associated value.

Options apply to both *ex* and *vi* modes, unless otherwise specified.

altwerase [off]
Vi only. Select an alternate word erase algorithm.

autoindent, ai [off]
Automatically indent new lines.

autoprint, ap [off]
Ex only. Display the current line automatically.

autowrite, aw 1[off]
Write modified files automatically when changing files.

backup [""] Backup files before they are overwritten.

beautify, bf [off]
Discard control characters.

cdpath [environment variable *CDPATH*, or current directory]
The directory paths used as path prefixes for the **cd** command.

cedit [no default]

Set the character to edit the colon command-line history.

columns, co [80]
Set the number of columns in the screen.

comment [off]
Vi only. Skip leading comments in shell and C-language files.

directory, **dir** [environment variable *TMPDIR*, or **/tmp**]
The directory where temporary files are created.

edcompatible, ed [off]
Remember the values of the "c" and "g" suffices to the **substitute** commands, instead of initializing them as unset for each new command.

errorbells, eb [off]
Ex only. Announce error messages with a bell.

exrc, ex [off] Read the startup files in the local directory.

extended [off]
Regular expressions are extended (i.e. *egrep*–style) expressions.

filec [no default]
Set the character to perform file path completion on the colon command line.

flash [on] Flash the screen instead of beeping the keyboard on error.

hardtabs, ht [8]
Set the spacing between hardware tab settings.

iclower [off] Makes all Regular Expressions case-insensitive, as long as an upper-case letter does not appear in the search string.

ignorecase, ic [off]
Ignore case differences in regular expressions.

keytime [6] The 10th's of a second *ex* / *vi* waits for a subsequent key to complete a key mapping.

leftright [off]
Vi only. Do left-right scrolling.

lines, li [24] *Vi* only. Set the number of lines in the screen.

lisp [off] *Vi* only. Modify various search commands and options to work with Lisp. *This option is not yet implemented.*

list [off] Display lines in an unambiguous fashion.

lock [on] Attempt to get an exclusive lock on any file being edited, read or written.

magic [on] Treat certain characters specially in regular expressions.

matchtime [7]
Vi only. The 10th's of a second *ex* / *vi* pauses on the matching character when the **showmatch** option is set.

mesg [on] Permit messages from other users.

modelines, modeline [off]
Read the first and last few lines of each file for *ex* commands. *This option will never be implemented.*

noprint [""] Characters that are never handled as printable characters.

number, nu [off]
Precede each line displayed with its current line number.

octal [off] Display unknown characters as octal numbers, instead of the default hexadecimal.

open [on] *Ex* only. If this option is not set, the **open** and **visual** commands are disallowed.

optimize, opt [on]
Vi only. Optimize text throughput to dumb terminals. *This option is not yet implemented.*
paragraphs, **para** [IPLPPPQPP LIpplpipbp]
Vi only. Define additional paragraph boundaries for the **{** and **}** commands.
path [] Define additional directories to search for files being edited.
print [""] Characters that are always handled as printable characters.
prompt [on] *Ex* only. Display a command prompt.
readonly, **ro** [off]
Mark the file and session as read-only.
recdir [**$OPENNT_ROOT/tmp/vi.recover**]
The directory where recovery files are stored.
redraw, re [off]
Vi only. Simulate an intelligent terminal on a dumb one. *This option is not yet implemented.*
remap [on] Remap keys until resolved.
report [5] Set the number of lines about which the editor reports changes or yanks.
ruler [off] *Vi* only. Display a row / column ruler on the colon command line.
scroll, scr [window / 2]
Set the number of lines scrolled.
searchincr [off]
Makes the **/** and **?** commands incremental.
sections, **sect** 1[NHSHH HUnhsh]
Vi only. Define additional section boundaries for the **[[** and **]]** commands.
secure [off] Turns off all access to external programs.
shell, **sh** [environment variable *SHELL*, or **/bin/sh**]
Select the shell used by the editor.
shellmeta [~{[*?$‘'"\]
Set the meta characters checked to determine if file name expansion is necessary.
shiftwidth, sw [8]
Set the autoindent and shift command indentation width.
showmatch, sm [off]
Vi only. Note matching “{” and “(” for “}” and “)” characters.
showmode, smd [off]
Vi only. Display the current editor mode and a “modified” flag.
sidescroll [16]
Vi only. Set the amount a left-right scroll will shift.
slowopen, slow [off]
Delay display updating during text input. *This option is not yet implemented.*
sourceany [off]
Read startup files not owned by the current user. *This option will never be implemented.*
tabstop, ts [8]
This option sets tab widths for the editor display.
taglength, tl [0]
Set the number of significant characters in tag names.

tags, tag [**tags**]
Set the list of tags files.
term, **ttytype**, **tty** [environment variable *TERM*]
Set the terminal type.
terse [off] This option has historically made editor messages less verbose. It has no effect in this implementation.
tildeop [off] Modify the ~ command to take an associated motion.
timeout, to [on]
Time out on keys which may be mapped.
ttywerase [off]
Vi only. Select an alternate erase algorithm.
verbose [off]
Vi only. Display an error message for every error.
w300 [no default]
Vi only. Set the window size if the baud rate is less than 1200 baud.
w1200 [no default]
Vi only. Set the window size if the baud rate is equal to 1200 baud.
w9600 [no default]
Vi only. Set the window size if the baud rate is greater than 1200 baud.
warn [on] *Ex* only. This option causes a warning message to the terminal if the file has been modified, since it was last written, before a **!** command.
window, **w**, **wi** [environment variable *LINES*]
Set the window size for the screen.
windowname [off]
Change the icon / window name to the current file name even if it can't be restored on editor exit.
wraplen, wl [0]
Vi only. Break lines automatically, the specified number of columns from the left-hand margin. If both the **wraplen** and **wrapmargin** edit options are set, the **wrapmargin** value is used.
wrapmargin, wm [0]
Vi only. Break lines automatically, the specified number of columns from the right-hand margin. If both the **wraplen** and **wrapmargin** edit options are set, the **wrapmargin** value is used.
wrapscan, ws [on]
Set searches to wrap around the end or beginning of the file.
writeany, **wa** [off]
Turn off file-overwriting checks.

DIAGNOSTICS

Ex/vi exits 0 on success, and greater than 0 if an error occurs.

ENVIRONMENTAL VARIABLES

COLUMNS
The number of columns on the screen. This value overrides any system or terminal specific values. If the *COLUMNS* environmental variable is not set when *ex* / *vi* runs, or the **columns** option is explicitly reset by the user, *ex* / *vi* enters the value into the environment.

EXINIT A list of *ex* startup commands, read if the variable *NEXINIT* is not set.

HOME The user's home directory, used as the initial directory path for the startup "$*HOME*/.nexrc" and "$*HOME*/.exrc" files. This value is also used as the default directory for the *vi* **cd** command.

LINES The number of rows on the screen. This value overrides any system or terminal specific values. If the *LINES* environmental variable is not set when *ex* / *vi* runs, or the **lines** option is explicitly reset by the user, *ex* / *vi* enters the value into the environment.

NEXINIT
A list of *ex* startup commands.

SHELL The user's shell of choice (see also the **shell** option).

TERM The user's terminal type. The default is the type "unknown". If the *TERM* environmental variable is not set when *ex* / *vi* runs, or the **term** option is explicitly reset by the user, *ex* / *vi* enters the value into the environment.

TMPDIR
The location used to stored temporary files (see also the **directory** edit option).

ASYNCHRONOUS EVENTS

SIGALRM *Vi* / *ex* uses this signal for periodic backups of file modifications and to display "busy" messages when operations are likely to take a long time.

SIGHUP
SIGTERM If the current buffer has changed since it was last written in its entirety, the editor attempts to save the modified file so it can be later recovered. See the *vi* / *ex* Reference manual section entitled "Recovery" for more information.

SIGINT When an interrupt occurs, the current operation is halted, and the editor returns to the command level. If interrupted during text input, the text already input is resolved into the file as if the text input had been normally terminated.

SIGWINCH The screen is resized. See the *vi* / *ex* Reference manual section entitled "Sizing the Screen" for more information.

SIGCONT
SIGQUIT
SIGTSTP *Vi* / *ex* ignores these signals.

FILES

Vi makes use of the following files:

/bin/sh The default user shell.

/etc/vi.exrc
System-wide *vi* startup file.

$OPENNT_ROOT/tmp
Temporary file directory, used if *TMPDIR* is not defined.

$OPENNT_ROOT/tmp/vi.recover
The default recovery file directory.

$HOME/.nexrc
1st choice for user's home directory startup file.

$HOME/.exrc
2nd choice for user's home directory startup file.

.nexrc 1st choice for local directory startup file.
.exrc 2nd choice for local directory startup file.

SEE ALSO

ctags, *more*

curses

The "Vi Quick Reference" card.

"An Introduction to Display Editing with Vi", found in the user manual of both the 4.3BSD and 4.4BSD manual sets. This document is the closest thing available to an introduction to the *vi* screen editor.

"Ex Reference Manual (Version 3.7)", found in the user manual of both the 4.3BSD and 4.4BSD manual sets. This document is the final reference for the *ex* editor, as distributed in most historic 4BSD and System V systems.

"Edit: A tutorial", found in the user manual of the 4.3BSD manual set. This document is an introduction to a simple version of the *ex* screen editor.

"Ex / Vi Reference Manual", found in the user manual of the 4.4BSD manual set. This document is the final reference for the *nex/nvi* text editors, as distributed in 4.4BSD and 4.4BSD-Lite.

Roff source for all of these documents is distributed with *nex/nvi* in the **nvi/USD.doc** directory of the *nex/nvi* source code.

The files "autowrite", "input", "quoting" and "structures" found in the **nvi/docs/internals** directory of the *nex/nvi* source code.

NAME

viewres - graphical class browser for Xt

SYNOPSIS

`viewres [-`*`option ...`*`]`

DESCRIPTION

The *viewres* program displays a tree showing the widget class hierarchy of the Athena Widget Set. Each node in the tree can be expanded to show the resources that the corresponding class adds (i.e. does not inherit from its parent) when a widget is created. This application allows the user to visually examine the structure and inherited resources for the Athena Widget Set.

OPTIONS

Viewres accepts all of the standard toolkit command line options as well as the following:

–top *name*

This option specifies the name of the highest widget in the hierarchy to display. This is typically used to limit the display to a subset of the tree. The default is *Object*.

–variable

This option indicates that the widget variable names (as declared in header files) should be displayed in the nodes rather than the widget class name. This is sometimes useful to distinguish widget classes that share the same name (such as *Text*).

–vertical

This option indicates that the tree should be displayed top to bottom rather left to right.

VIEW MENU

The way in which the tree is displayed may be changed through the entries in the **View** menu:

Show Variable Names

This entry causes the node labels to be set to the variable names used to declare the corresponding widget class. This operation may also be performed with the **SetLabelType(variable)** translation.

Show Class Names

This entry causes the node labels to be set to the class names used when specifying resources. This operation may also be performed with the **SetLabelType(class)** translation.

Layout Horizontal

This entry causes the tree to be laid out from left to right. This operation may also be performed with the *SetOrientation(West)* translation.

Layout Vertical

This entry causes the tree to be laid out from top to bottom. This operation may also be performed with the *SetOrientation(North)* translation.

Show Resource Boxes
This entry expands the selected nodes (see next section) to show the new widget and constraint resources. This operation may also be performed with the *Resources(on)* translation.

Hide Resource Boxes
This entry removes the resource displays from the selected nodes (usually to conserve space). This operation may also be performed with the *Resources(off)* translation.

SELECT MENU

Resources for a single widget class can be displayed by clicking **Button2** on the corresponding node, or by adding the node to the selection list with **Button1** and using the **Show Resource Boxes** entry in the **View** menu. Since **Button1** actually toggles the selection state of a node, clicking on a selected node will cause it to be removed from the selected list.

Collections of nodes may also be selected through the various entries in the **Select** menu:

Unselect All
This entry removes all nodes from the selection list. This operation may also be performed with the *Select(nothing)* translation.

Select All
This entry adds all nodes to the selection list. This operation may also be performed with the *Select(all)* translation.

Invert All
This entry adds unselected nodes to, and removes selected nodes from, the selection list. This operation may also be performed with the *Select(invert)* translation.

Select Parent
This entry selects the immediate parents of all selected nodes. This operation may also be performed with the *Select(parent)* translation.

Select Ancestors
This entry recursively selects all parents of all selected nodes. This operation may also be performed with the *Select(ancestors)* translation.

Select Children
This entry selects the immediate children of all selected nodes. This operation may also be performed with the *Select(children)* translation.

Select Descendants
This entry recursively selects all children of all selected nodes. This operation may also be performed with the *Select(descendants)* translation.

Select Has Resources
This entry selects all nodes that add new resources (regular or constraint) to their corresponding widget classes. This operation may also be performed with the *Select(resources)* translation.

Select Shown Resource Boxes
This entry selects all nodes whose resource boxes are currently expanded (usually so that they can be closed with **Hide Resource Boxes**). This operation may also be performed with the *Select(shown)* translation.

ACTIONS

The following application actions are provided:

Quit() This action causes *viewres* to exit.

SetLabelType(*type***)**

This action sets the node labels to display the widget *variable* or *class* names, according to the argument *type*.

SetOrientation(*direction***)**

This action sets the root of the tree to be one of the following areas of the window: *West*, *North*, *East*, or *South*.

Select(*what***)**

This action selects the indicated nodes, as described in the **VIEW MENU** section: *nothing* (unselects all nodes), *invert*, *parent*, *ancestors*, *children*, *descendants*, *resources*, *shown*.

Resources(*op***)**

This action turns *on*, *off*, or *toggles* the resource boxes for the selected nodes. If invoked from within one of the nodes (through the keyboard or pointer), only that node is used.

WIDGET HIERARCHY

Resources may be specified for the following widgets:

```
Viewres viewres
   Paned pane
      Box buttonbox
         Command quit
         MenuButton view
            SimpleMenu viewMenu
               SmeBSB layoutHorizontal
               SmeBSB layoutVertical
               SmeLine line1
               SmeBSB namesVariable
               SmeBSB namesClass
               SmeLine line2
               SmeBSB viewResources
               SmeBSB viewNoResources
         MenuButton select
            SimpleMenu selectMenu
               SmeBSB unselect
               SmeBSB selectAll
               SmeBSB selectInvert
               SmeLine line1
               SmeBSB selectParent
               SmeBSB selectAncestors
               SmeBSB selectChildren
               SmeBSB selectDescendants
               SmeLine line2
               SmeBSB selectHasResources
               SmeBSB selectShownResources
      Form treeform
         Porthole porthole
            Tree tree Box variable-name
                  Toggle variable-name
```

```
                    List variable-name
              Panner panner
```

where *variable-name* is the widget variable name of each node.

SEE ALSO

X, *xrdb*, *listres*, *editres*, *appres*, appropriate widget documents

COPYRIGHT

Copyright 1990, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

wait – wait for process completion

SYNOPSIS

`wait [`*`jobid`*`]`

DESCRIPTION

The *wait* command waits for the specified job(s) to finish. The *jobid* is specified either by the process ID of the last process in the command pipeline, or by prefixing the job-number (revealed by the *jobs* command) with a percent sign (**%**).

Other special job IDs are:

%+	The most recently stopped job, or, if there are no stopped jobs, the oldest running job.
%%, **%**	Same as **%+**.
%–	The job that would be the **%+** job, if the later did not exist.
%*n*	The job with job-number *n*.
%?*string*	The job containing the string *string* (an error occurs if multiple jobs are matched).
%*string*	The job starting with string *string* (an error occurs if multiple jobs are matched).

Wait will return if a signal for which a trap has been set is received, or if a HUP, INT or QUIT signal is received.

If no jobs are specified, *wait* waits for all currently running jobs (if any) to finish and exits with a zero status. If job monitoring is enabled, the completion status of jobs is printed (this is not the case when jobs are explicitly specified).

Wait is a shell built-in command.

DIAGNOSTICS

Wait exits with the status of the last specified job, except:

- If the last job is killed by a signal, the exit status is 128+ the number of the signal.
- If the last job can't be found (it got killed or it had already finished), the exit status is 127.

SEE ALSO

sh

NAME

wc – word, line, and byte count

SYNOPSIS

```
wc [-c|-m [lw]] [ file ...]
```

DESCRIPTION

The *wc* utility displays the number of lines, words, and bytes contained in each input *file* (or standard input, by default) to the standard output. A line is defined as a string of characters delimited by a <newline> character, and a word is defined as a string of characters delimited by white space characters. White space characters are the set of characters for which the *isspace*( ) function returns true. If more than one input file is specified, a line of cumulative counts for all the files is displayed on a separate line after the output for the last file.

The following options are available:

–c The number of bytes in each input file is written to the standard output.

–l The number of lines in each input file is written to the standard output.

–m The number of characters in each input file is written to the standard output. In a multibyte environment, this may be different from the number of bytes.

–w The number of words in each input file is written to the standard output.

When an option is specified, *wc* only reports the information requested by that option. The default action is equivalent to specifying all of the flags.

If no files are specified, the standard input is used and no file name is displayed.

DIAGNOSTICS

The *wc* utility exits 0 on success, and >0 if an error occurs.

NAME

who – display who is logged in

SYNOPSIS

```
who [-mu] -s [-bfHlprt] [file]
who [-mTu] [-abdfHlprt] [file]
who -q [file]
who am i
who am I
```

DESCRIPTION

The utility *who* displays information about users currently logged in to the machine. Without options, *who* shows for each user the login name, terminal line, and login time. Options display additional information.

The last two forms,

```
who am i
who am I
```

are equivalent to

```
who -m
```

By default, *who* gathers information from the file **$OPENNT_ROOT/var/adm/utmpx**. You can specify an alternate *file* (usually **$OPENNT_ROOT/var/adm/wtmpx**) as the source of the information.

The general format of the output is:

name **[***state***]** *line time* **[***activity***][***pid***][***host***]**

Where the values are:

name	The user's login name.
state	The state of the terminal; see the description of the **–T** option.
line	The number of the terminal line (the file under **/dev**).
time	Login time.
activity	The number of hours and minutes since activity last occurred. A dot (.) indicates activity within the last minute; the string "<old>" indicates that no activity has occurred for 24 hours.
pid	The process ID of the user's login process.
host	The name of the host computer, if **–f** is specified.
exit	The exit status of dead processes.

Available options:

–a Display all utmpx record types; this is similar to the union of options **–bdprt**. Options that control control how things are displayed aren't affected.

–b Display time and date of last system reboot.

–d Display list of dead processes (processes that have expired). This information is displayed in the *exit* field of the output. On traditional systems, this option lists processes that have not been respawned by *init*; since INTERIX does not have an *init* process, **–d** lists only dead processes.

–f Display the name of the host as the last field in the output.

–H Write column headings above the regular output.

–l (The letter ell.) Display terminal lines waiting for someone to login. The *name* field is displayed as LOGIN. This information is not currently stored in the INTERIX subsystem, so this option is not yet useful.

–m Return information about the current terminal only.

–p List any other process that is currently active and that was spawned by *init*. Since INTERIX has no *init* process, this option is not useful.

–q Quick output: list only the names and number of users currently logged on. When this option is used, all other options are ignored.

–r Display current run-level of the *init* process. Since INTERIX has no *init* process, this option is not useful.

–s List only the *name*, *line*, and *time* fields. This is the default behavior for *who*.

–t Display the last change to the system clock. This information is not currently stored in the INTERIX subsystem, so this option is not yet useful.

–T Display the message state of each terminal; a terminal state is indicated by the characters **+**, **-**, **?**:

- **+** Other users can send messages to the terminal.
- **-** Other users cannot send messages to the terminal.
- **?** The state cannot be determined.

–u Displays “idle time” for each user; idle time is the time since any activity occurred at the user’s terminal.

DIAGNOSTICS

Who returns 0 when it succeeds, and >0 if an error occurs.

FILES

The following files are used by *who*:

$OPENNT_ROOT/var/adm/utmpx
A database of system users.

$OPENNT_ROOT/var/adm/wtmpx
Database of the history of system users.

SEE ALSO

whoami

NAME

whoami – display effective user id

SYNOPSIS

```
whoami [-s]
```

DESCRIPTION

The *whoami* utility displays your effective user ID as a Windows NT domain name and a login name, separated by a + sign. For example:

```
$ whoami
dev+jcat
```

where "dev" is the domain and "jcat" is the login name. This is equivalent to the value of the shell variable *LOGNAME*.

The option available for *whoami* is:

–s Strip domain name from output. This is equivalent to the value of the shell variable *USERNAME*.

DIAGNOSTICS

The *whoami* utility exits 0 on success, and >0 if an error occurs.

NOTES

The *whoami* utility has been obsoleted by the *id* utility. It is provided here as a shell script, equivalent to

```
id -un
```

The command

```
id -p
```

is suggested for normal interactive use.

SEE ALSO

id

NAME

xargs – construct argument list(s) and execute utility

SYNOPSIS

```
xargs [-tx] [-n number] [-s size] [utility
      [arguments ...]]
```

DESCRIPTION

The *xargs* utility reads arguments from standard input and then runs the specified *utility* with those arguments (and any specified on the command line). Arguments are delimited by spaces, tabs, newlines, and end-of-file. *Xargs* is often used for splitting very long argument lists into smaller chunks.

The utility and any arguments specified on the command line are given to the *utility* upon each invocation, followed by some number of the arguments read from standard input. The *utility* is repeatedly executed until standard input is exhausted.

Spaces, tabs and newlines may be embedded in arguments using single quotes (’). Single quotes escape all non-single quote characters, excluding newlines, up to the matching single quote. Double quotes escape all non-double quote characters, excluding newlines, up to the matching double quote. Any single character, including newlines, may be escaped by a backslash.

The options are as follows:

–n *number*
: Set the maximum number of arguments taken from standard input for each invocation of the utility. An invocation of *utility* will use less than *number* standard input arguments if the number of bytes accumulated (see the **–s** option) exceeds the specified *size* or there are fewer than *number* arguments remaining for the last invocation of *utility*. The current default value for *number* is 5000.

–s *size*
: Set the maximum number of bytes for the command line length provided to *utility*. The sum of the length of the utility name and the arguments passed to *utility* (including NULL terminators) will be less than or equal to this number. The current default value for *size* is {ARG_MAX} –4096.

–t
: Echo the command to be executed to standard error immediately before it is executed.

–x
: Force *xargs* to terminate immediately if a command line containing *number* arguments will not fit in the specified (or default) command line length.

If no *utility* is specified, *xargs* executes *echo*.

Undefined behavior may occur if *utility* reads from the standard input.

The *xargs* utility exits immediately (without processing any further input) if a command line cannot be assembled, *utility* cannot be invoked, an invocation of the utility is terminated by a signal or an invocation of the utility exits with a value of 255.

DIAGNOSTICS

The *xargs* utility exits with a value of 0 if no error occurs. If *utility* cannot be invoked, *xargs* exits with a value of 127. If any other error occurs, *xargs* exits with a value of 1.

EXAMPLES

Run the awk script **~/lib/convert.awk** on all files in the current directory (assuming there are too many to use * as the file specifier):

```
ls | xargs awk -f ~/lib/convert.awk
```

Remove all files that haven't been modified in at least four weeks:

```
find . -type f -mtime +28 -print | xargs rm
```

SEE ALSO

echo, *find*

NAME

X – a portable, network-transparent window system

SYNOPSIS

The X Window System is a network transparent window system developed at MIT which runs on a wide range of computing and graphics machines. It should be relatively straightforward to build the MIT software distribution on most ANSI C and POSIX compliant systems. Commercial implementations are also available for a wide range of platforms.

The X Consortium requests that the following names be used when referring to this software:

X
X Window System
X Version 11
X Window System, Version 11
X11

X Window System is a trademark of the Massachusetts Institute of Technology.

DESCRIPTION

X Window System servers run on computers with bitmap displays. The server distributes user input to and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of the functions that are available, see the *Xlib — C Language X Interface* manual, the *X Window System Protocol* specification, the *X Toolkit Intrinsics — C Language Interface* manual, and various toolkit documents.

The number of programs that use X is quite large. Programs provided in the core MIT distribution include: a terminal emulator (*xterm*), a window manager (*twm*), a display manager (*xdm*), a console redirect program (*xconsole*), mail managing utilities (*xmh* and *xbiff*), a manual page browser (*xman*), a bitmap editor (*bitmap*), a resource editor (*editres*), a ditroff previewer (*xditview*), access control programs (*xauth* and *xhost*), user preference setting programs (*xrdb*, *xcmsdb*, *xset*, *xsetroot*, *xstdcmap*, and *xmodmap*), a load monitor (*xload*), clocks (*xclock* and *oclock*), a font displayer (*xfd*), utilities for listing information about fonts, windows, and displays (*xlsfonts*, *xfontsel*, *xwininfo*, *xlsclients*, *xdpyinfo*, and *xprop*), a diagnostic for seeing what events are generated and when (*xev*), screen image manipulation utilities (*xwd*, *xwud*, *xpr*, and *xmag*), and various demos (*xeyes*, *ico*, *xgc*, *x11perf*, etc.).

Many other utilities, window managers, games, toolkits, etc. are included as user-contributed software in the MIT distribution, or are available using anonymous ftp on the Internet. See your site administrator for details.

STARTING UP

There are two main ways of getting the X server and an initial set of client applications started. The particular method used depends on what operating system you are running and on whether or not you use other window systems in addition to X.

xdm (the X Display Manager)
: If you want to always have X running on your display, your site administrator can set your machine up to use the X Display Manager *xdm*. This program is typically started by the system at boot time and takes care of keeping the server running and getting users logged in. If you are running *xdm*, you will see a window on the screen welcoming you to the system and asking for your username and password. Simply type them in as you would at a normal terminal, pressing the Return key after each. If you make a mistake, *xdm* will display an error message and ask you to try again. After you have successfully logged in, *xdm* will start up your X environment. By default, if you have an executable file named **.xsession** in your home directory, *xdm* will treat it as a program (or shell script) to run to start up your initial clients (such as terminal emulators, clocks, a window manager, user settings for things like the background, the speed of the pointer, etc.). Your site administrator can provide details.

xinit (run manually from the shell)
: Sites that support more than one window system might choose to use the *xinit* program for starting X manually. If this is true for your machine, your site administrator will probably have provided a program named "x11", "startx", or "xstart" that will do site-specific initialization (such as loading convenient default resources, running a window manager, displaying a clock, and starting several terminal emulators) in a nice way. If not, you can build such a script using the *xinit* program. This utility simply runs one user-specified program to start the server, runs another to start up any desired clients, and then waits for either to finish. Since either or both of the user-specified programs may be a shell script, this gives substantial flexibility at the expense of a nice interface. For this reason, *xinit* is not intended for end users.

DISPLAY NAMES

From the user's prospective, every X server has a *display name* of the form:

`hostname:displaynumber.screennumber`

This information is used by the application to determine how it should connect to the server and which screen it should use by default (on displays with multiple monitors):

hostname
: The *hostname* specifies the name of the machine to which the display is physically connected. If the hostname is not given, the most efficient way of communicating to a server on the same machine will be used.

displaynumber
: The phrase "display" is usually used to refer to collection of monitors that share a common keyboard and pointer (mouse, tablet, etc.). Most

workstations tend to only have one keyboard, and therefore, only one display. Larger, multi-user systems, however, will frequently have several displays so that more than one person can be doing graphics work at once. To avoid confusion, each display on a machine is assigned a *display number* (beginning at 0) when the X server for that display is started. The display number must always be given in a display name.

screennumber

Some displays share a single keyboard and pointer among two or more monitors. Since each monitor has its own set of windows, each screen is assigned a *screen number* (beginning at 0) when the X server for that display is started. If the screen number is not given, then screen 0 will be used.

On POSIX systems, the default display name is stored in your *DISPLAY* environment variable. This variable is set automatically by the *xterm* terminal emulator. However, when you log into another machine on a network, you'll need to set *DISPLAY* by hand to point to your display. For example,

```
% setenv DISPLAY myws:0
$ DISPLAY=myws:0; export DISPLAY
```

The *xon* script can be used to start an X program on a remote machine; it automatically sets the *DISPLAY* variable correctly.

Finally, most X programs accept a command line option of **–display** *displayname* to temporarily override the contents of *DISPLAY*. This is most commonly used to pop windows on another person's screen or as part of a "remote shell" command to start an xterm pointing back to your display. For example,

```
% xeyes -display joesws:0 -geometry 1000x1000+0+0
% rsh big xterm -display myws:0 -ls </dev/null &
```

X servers listen for connections on a variety of different communications channels (network byte streams, shared memory, etc.). Since there can be more than one way of contacting a given server, The *hostname* part of the display name is used to determine the type of channel (also called a transport layer) to be used. X servers generally support the following types of connections:

local The hostname part of the display name should be the empty string. For example: *:0*, *:1*, and *:0.1*. The most efficient local transport will be chosen.

TCP/IP The hostname part of the display name should be the server machine's IP address name. Full Internet names, abbreviated names, and IP addresses are all allowed. For example: *expo.lcs.mit.edu:0*, *expo:0*, *18.30.0.212:0*, *bigmachine:1*, and *hydra:0.1*.

DECnet The hostname part of the display name should be the server machine's nodename followed by two colons instead of one. For example: *myws::0*, *big::1*, and *hydra::0.1*.

ACCESS CONTROL

An X server can use several types of access control. Mechanisms provided in Release 5 are:

Host Access	Simple host-based access control.
MIT-MAGIC-COOKIE-1	Shared plain-text "cookies".
XDM-AUTHORIZATION-1	Secure DES based private-keys.
SUN-DES-1	Based on Sun's secure rpc system.

Xdm initializes access control for the server, and also places authorization information in a file accessible to the user. Normally, the list of hosts from which connections are always accepted should be empty, so that only clients with are explicitly authorized can connect to the display. When you add entries to the host list (with *xhost*), the server no longer performs any authorization on connections from those machines. Be careful with this.

The file from which the Xlib extracts authorization data can be specified with the environment variable *XAUTHORITY*, and defaults to the file **.Xauthority** in the home directory. *Xdm* uses **$HOME/.Xauthority** and will create it or merge in authorization records if it already exists when a user logs in.

If you use several machines, and share a common home directory across all of the machines by means of a network file system, then you never really have to worry about authorization files, the system should work correctly by default. Otherwise, as the authorization files are machine-independent, you can simply copy the files to share them. To manage authorization files, use *xauth*. This program allows you to extract records and insert them into other files. Using this, you can send authorization to remote machines when you login, if the remote machine does not share a common home directory with your local machine. Note that authorization information transmitted ''in the clear'' through a network file system or using *ftp* or *rcp* can be ''stolen'' by a network eavesdropper, and as such may enable unauthorized access. In many environments this level of security is not a concern, but if it is, you need to know the exact semantics of the particular authorization data to know if this is actually a problem.

For more information on access control, see the *Xsecurity* manual page.

GEOMETRY SPECIFICATIONS

One of the advantages of using window systems instead of hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running (described below), most X programs accept a command line argument of the form **–geometry** *WIDTH***x***HEIGHT*+*XOFF*+*YOFF* (where *WIDTH, HEIGHT, XOFF*, and *YOFF* are numbers) for specifying a preferred size and location for this application's main window.

The *WIDTH* and *HEIGHT* parts of the geometry specification are usually measured in either pixels or characters, depending on the application. The *XOFF* and *YOFF* parts are measured in pixels and are used to specify the distance of the window from the left or right and top and bottom edges of the screen, respectively. Both types of offsets are measured from the indicated edge of the screen to the corresponding edge of the window. The X offset may

be specified in the following ways:

+XOFF The left edge of the window is to be placed *XOFF* pixels in from the left edge of the screen (i.e. the X coordinate of the window's origin will be *XOFF*). *XOFF* may be negative, in which case the window's left edge will be off the screen.

–XOFF The right edge of the window is to be placed *XOFF* pixels in from the right edge of the screen. *XOFF* may be negative, in which case the window's right edge will be off the screen.

The Y offset has similar meanings:

+YOFF The top edge of the window is to be *YOFF* pixels below the top edge of the screen (i.e. the Y coordinate of the window's origin will be *YOFF*). *YOFF* may be negative, in which case the window's top edge will be off the screen.

–YOFF The bottom edge of the window is to be *YOFF* pixels above the bottom edge of the screen. *YOFF* may be negative, in which case the window's bottom edge will be off the screen.

Offsets must be given as pairs; in other words, in order to specify either *XOFF* or *YOFF* both must be present. Windows can be placed in the four corners of the screen using the following specifications:

+0+0 upper left hand corner.
-0+0 upper right hand corner.
-0-0 lower right hand corner.
+0-0 lower left hand corner.

In the following examples, a terminal emulator will be placed in roughly the center of the screen and a load average monitor, mailbox, and clock will be placed in the upper right hand corner:

```
xterm -fn 6x10 -geometry 80x24+30+200 &
xclock -geometry 48x48-0+0 &
xload -geometry 48x48-96+0 &
xbiff -geometry 48x48-48+0 &
```

WINDOW MANAGERS

The layout of windows on the screen is controlled by special programs called *window managers*. Although many window managers will honor geometry specifications as given, others may choose to ignore them (requiring the user to explicitly draw the window's region on the screen with the pointer, for example).

Since window managers are regular (albeit complex) client programs, a variety of different user interfaces can be built. The MIT distribution comes with a window manager named *twm* which supports overlapping windows, popup menus, point-and-click or click-to-type input models, title bars, nice icons (and an icon manager for those who don't like separate icon windows).

See the user-contributed software in the MIT distribution for other popular window managers.

FONT NAMES

Collections of characters for displaying text and symbols in X are known as *fonts*. A font typically contains images that share a common appearance and

look nice together (for example, a single size, boldness, slant, and character set). Similarly, collections of fonts that are based on a common type face (the variations are usually called roman, bold, italic, bold italic, oblique, and bold oblique) are called *families*.

Fonts come in various sizes. The X server supports *scalable* fonts, meaning it is possible to create a font of arbitrary size from a single source for the font. The server supports scaling from *outline* fonts and *bitmap* fonts. Scaling from outline fonts usually produces significantly better results than scaling from bitmap fonts.

An X server can obtain fonts from individual files stored in directories in the file system, or from one or more font servers, or from a mixtures of directories and font servers. The list of places the server looks when trying to find a font is controlled by its *font path*. Although most installations will choose to have the server start up with all of the commonly used font directories in the font path, the font path can be changed at any time with the *xset* program. However, it is important to remember that the directory names are on the **server**'s machine, not on the application's. The most common fonts use by X servers and font servers can be found in four directories:

$OPENNT_ROOT/usr/lib/X11/fonts/misc
: This directory contains many miscellaneous bitmap fonts that are useful on all systems. It contains a family of fixed-width fonts, a family of fixed-width fonts from Dale Schumacher, several Kana fonts from Sony Corporation, two JIS Kanji fonts, two Hangul fonts from Daewoo Electronics, two Hebrew fonts from Joseph Friedman, the standard cursor font, two cursor fonts from Digital Equipment Corporation, and cursor and glyph fonts from Sun Microsystems. It also has various font name aliases for the fonts, including **fixed** and **variable**.

$OPENNT_ROOT/usr/lib/X11/fonts/Speedo
: This directory contains outline fonts for Bitstream's Speedo rasterizer. A single font face, in normal, bold, italic, and bold italic, is provided, contributed by Bitstream, Inc.

$OPENNT_ROOT/usr/lib/X11/fonts/75dpi
: This directory contains bitmap fonts contributed by Adobe Systems, Inc., Digital Equipment Corporation, Bitstream, Inc., Bigelow and Holmes, and Sun Microsystems, Inc. for 75 dots per inch displays. An integrated selection of sizes, styles, and weights are provided for each family.

$OPENNT_ROOT/usr/lib/X11/fonts/100dpi
: This directory contains 100 dots per inch versions of some of the fonts in the **75dpi** directory.

Bitmap font files are usually created by compiling a textual font description into binary form, using *bdftopcf*. Font databases are created by running the *mkfontdir* program in the directory containing the source or compiled versions of the fonts. Whenever fonts are added to a directory, *mkfontdir* should be rerun so that the server can find the new fonts. To make the server reread the font database, reset the font path with the *xset* program. For example, to add a font to a private directory, the following commands could be used:

```
% cp newfont.pcf ~/myfonts
% mkfontdir ~/myfonts
% xset fp rehash
```

The *xfontsel* and *xlsfonts* programs can be used to browse through the fonts available on a server. Font names tend to be fairly long as they contain all of the information needed to uniquely identify individual fonts. However, the X server supports wildcarding of font names, so the full specification

```
-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1
```

might be abbreviated as:

```
-*-courier-medium-r-normal--*-100-*-*-*-*-iso8859-1
```

Because the shell also has special meanings for * and *?*, wildcarded font names should be quoted:

```
% xlsfonts -fn '-*-courier-medium-r-normal--*-100-*-*-*-*-*-*'
```

The *xlsfonts* program can be used to list all of the fonts that match a given pattern. With no arguments, it lists all available fonts. This will usually list the same font at many different sizes. To see just the base scalable font names, try using one of the following patterns:

```
-*-*-*-*-*-*-0-0-0-0-*-0-*-*
-*-*-*-*-*-*-0-0-75-75-*-0-*-*
-*-*-*-*-*-*-0-0-100-100-*-0-*-*
```

To convert one of the resulting names into a font at a specific size, replace one of the first two zeros with a nonzero value. The field containing the first zero is for the pixel size; replace it with a specific height in pixels to name a font at that size. Alternatively, the field containing the second zero is for the point size; replace it with a specific size in decipoints (there are 722.7 decipoints to the inch) to name a font at that size. The last zero is an average width field, measured in tenths of pixels; some servers will anamorphically scale if this value is specified.

FONT SERVER NAMES

One of the following forms can be used to name a font server that accepts TCP connections:

```
tcp/hostname:port
tcp/hostname:port/cataloguelist
```

The *hostname* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *port* is the decimal TCP port on which the font server is listening for connections. The *cataloguelist* specifies a list of catalogue names, with '+' as a separator.

Examples: *tcp/expo.lcs.mit.edu:7000*, *tcp/18.30.0.212:7001/all*.

One of the following forms can be used to name a font server that accepts DECnet connections:

```
decnet/nodename::font$objname
decnet/nodename::font$objname/cataloguelist
```

The *nodename* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *objname* is a normal, case-insensitive DECnet object name. The *cataloguelist* specifies a list of catalogue names, with '+' as a separator.

Examples: *DECnet/SRVNOD::FONT$DEFAULT*, *decnet/44.70::font$special/symbols*.

COLOR NAMES

Most applications provide ways of tailoring (usually through resources or command line arguments) the colors of various elements in the text and graphics they display. A color can be specified either by an abstract color name, or by a numerical color specification. The numerical specification can identify a color in either device-dependent (RGB) or device-independent terms. Color strings are case-insensitive.

X supports the use of abstract color names, for example, "red", "blue". A value for this abstract name is obtained by searching one or more color name databases. *Xlib* first searches zero or more client-side databases; the number, location, and content of these databases is implementation dependent. If the name is not found, the color is looked up in the X server's database. The text form of this database is commonly stored in the file *$OPENNT_ROOT/usr/lib/X11/rgb.txt*.

A numerical color specification consists of a color space name and a set of values in the following syntax:

```
color_space_name:value/.../value
```

An RGB Device specification is identified by the prefix "rgb:" and has the following syntax:

```
rgb:red/green/blue

red, green, blue := h | hh | hhh | hhhh
h := single hexadecimal digits
```

Note that *h* indicates the value scaled in 4 bits, *hh* the value scaled in 8 bits, *hhh* the value scaled in 12 bits, and *hhhh* the value scaled in 16 bits, respectively. These values are passed directly to the X server, and are assumed to be gamma corrected.

black	`rgb:0/0/0`
red	`rgb:ffff/0/0`
green	`rgb:0/ffff/0`
blue	`rgb:0/0/ffff`
yellow	`rgb:ffff/ffff/0`
magenta	`rgb:ffff/0/ffff`
cyan	`rgb:0/ffff/ffff`
white	`rgb:ffff/ffff/ffff`

For backward compatibility, an older syntax for RGB Device is supported, but its continued use is not encouraged. The syntax is an initial sharp sign character followed by a numeric specification, in one of the following formats:

```
#RGB              (4 bits each)
#RRGGBB           (8 bits each)
#RRRGGGBBB        (12 bits each)
#RRRRGGGGBBBB     (16 bits each)
```

The R, G, and B represent single hexadecimal digits. When fewer than 16 bits each are specified, they represent the most-significant bits of the value (unlike the "rgb:" syntax, in which values are scaled). For example, #3a7 is the same as #3000a0007000.

An RGB intensity specification is identified by the prefix "rgbi:" and has the following syntax:

`rgbi:`*`red/green/blue`*

The red, green, and blue are floating point values between 0.0 and 1.0, inclusive. They represent linear intensity values, with 1.0 indicating full intensity, 0.5 half intensity, and so on. These values will be gamma corrected by *Xlib* before being sent to the X server. The input format for these values is an optional sign, a string of numbers possibly containing a decimal point, and an optional exponent field containing an E or e followed by a possibly signed integer string.

The standard device-independent string specifications have the following syntax:

`CIEXYZ:`*`X/Y/Z`*	(*none*, 1, *none*)
`CIEuvY:`*`u/v/Y`*	(˜.6, ˜.6, 1)
`CIExyY:`*`x/y/Y`*	(˜.75, ˜.85, 1)
`CIELab:`*`L/a/b`*	(100, *none*, *none*)
`CIELuv:`*`L/u/v`*	(100, *none*, *none*)
`TekHVC:`*`H/V/C`*	(360, 100, 100)

All of the values (C, H, V, X, Y, Z, a, b, u, v, y, x) are floating point values. Some of the values are constrained to be between zero and some upper bound; the upper bounds are given in parentheses above. The syntax for these values is an optional '+' or '-' sign, a string of digits possibly containing a decimal point, and an optional exponent field consisting of an 'E' or 'e' followed by an optional '+' or '-' followed by a string of digits.

For more information on device independent color, see the *Xlib* reference manual.

KEYBOARDS

The X keyboard model is broken into two layers: server-specific codes (called *keycodes*) which represent the physical keys, and server-independent symbols (called *keysyms*) which represent the letters or words that appear on the keys. Two tables are kept in the server for converting keycodes to keysyms:

modifier list

Some keys (such as Shift, Control, and Caps Lock) are known as *modifier* and are used to select different symbols that are attached to a single key (such as Shift-a generates a capital A, and Control-l generates a control character ^L). The server keeps a list of keycodes corresponding to the various modifier keys. Whenever a key is

pressed or released, the server generates an *event* that contains the keycode of the indicated key as well as a mask that specifies which of the modifier keys are currently pressed. Most servers set up this list to initially contain the various shift, control, and shift lock keys on the keyboard.

keymap table

Applications translate event keycodes and modifier masks into keysyms using a *keysym table* which contains one row for each keycode and one column for various modifier states. This table is initialized by the server to correspond to normal typewriter conventions. The exact semantics of how the table is interpreted to produce keysyms depends on the particular program, libraries, and language input method used, but the following conventions for the first four keysyms in each row are generally adhered to:

The first four elements of the list are split into two groups of keysyms. Group 1 contains the first and second keysyms; Group 2 contains the third and fourth keysyms. Within each group, if the first element is alphabetic and the the second element is the special keysym *NoSymbol*, then the group is treated as equivalent to a group in which the first element is the lowercase letter and the second element is the uppercase letter.

Switching between groups is controlled by the keysym named *MODE SWITCH*, by attaching that keysym to some key and attaching that key to any one of the modifiers Mod1 through Mod5. This modifier is called the ''group modifier.'' Group 1 is used when the group modifier is off, and Group 2 is used when the group modifier is on.

Within a group, the modifier state determines which keysym to use. The first keysym is used when the Shift and Lock modifiers are off. The second keysym is used when the Shift modifier is on, when the Lock modifier is on and the second keysym is uppercase alphabetic, or when the Lock modifier is on and is interpreted as ShiftLock. Otherwise, when the Lock modifier is on and is interpreted as CapsLock, the state of the Shift modifier is applied first to select a keysym; but if that keysym is lowercase alphabetic, then the corresponding uppercase keysym is used instead.

OPTIONS

Most X programs attempt to use the same names for command line options and arguments. All applications written with the X Toolkit Intrinsics automatically accept the following options:

–display *display*

This option specifies the name of the X server to use.

–geometry *geometry*

This option specifies the initial size and location of the window.

"**–bg** *color*, **–background** *color*

Either option specifies the color to use for the window background.

–bd *color*, **–bordercolor** *color*

Either option specifies the color to use for the window border.

–bw *number*, **–borderwidth** *number*

Either option specifies the width in pixels of the window border.

–fg *color*, **–foreground** *color*

Either option specifies the color to use for text or graphics.

–fn *font*, **-font** *font*
Either option specifies the font to use for displaying text.

–iconic This option indicates that the user would prefer that the application's windows initially not be visible as if the windows had be immediately iconified by the user. Window managers may choose not to honor the application's request.

–name This option specifies the name under which resources for the application should be found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.

–rv, **–reverse**
Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.

+rv This option indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.

–selectionTimeout
This option specifies the timeout in milliseconds within which two communicating applications must respond to one another for a selection request.

–synchronous
This option indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since *Xlib* normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.

–title *string*
This option specifies the title to be used for this window. This information is sometimes used by a window manager to provide some sort of header identifying the window.

–xnllanguage *language[_territory][.codeset]*
This option specifies the language, territory, and codeset for use in resolving resource and other filenames.

–xrm *resourcestring*
This option specifies a resource name and value to override any defaults. It is also very useful for setting resources that don't have explicit command line arguments.

RESOURCES

To make the tailoring of applications to personal preferences easier, X provides a mechanism for storing default values for program resources (e.g. background color, window title, etc.) Resources are specified as strings that are read in from various places when an application is run. Program components are named in a hierarchical fashion, with each node in the hierarchy identified by a class and an instance name. At the top level is the class and instance name of the application itself. By convention, the class name of the application is the same as the program name, but with the first letter capitalized (e.g. *Bitmap* or *Emacs*) although some programs that begin with the letter ''x'' also capitalize the second letter for historical reasons.

The precise syntax for resources is:

```
ResourceLine    = Comment | IncludeFile | ResourceSpec |
                  <empty line>
Comment         = "!" {<any character except null or newline>}
IncludeFile     = "#" WhiteSpace "include" WhiteSpace
                  FileName WhiteSpace
FileName        = <valid filename for operating system>
ResourceSpec    = WhiteSpace ResourceName WhiteSpace ":"
                  WhiteSpace Value
ResourceName    = [Binding] {Component Binding} ComponentName
Binding         = "." | "*"
WhiteSpace      = {<space> | <horizontal tab>}
Component       = "?" | ComponentName
ComponentName   = NameChar {NameChar}
NameChar        = "a"–"z" | "A"–"Z" | "0"–"9" | "_" | "-"
Value           = {<any character except null or unescaped newline>}
```

Elements separated by vertical bar (|) are alternatives. Curly braces ({...}) indicate zero or more repetitions of the enclosed elements. Square brackets ([...]) indicate that the enclosed element is optional. Quotes ("...") are used around literal characters.

IncludeFile lines are interpreted by replacing the line with the contents of the specified file. The word "include" must be in lowercase. The filename is interpreted relative to the directory of the file in which the line occurs (for example, if the filename contains no directory or contains a relative directory specification).

If a ResourceName contains a contiguous sequence of two or more Binding characters, the sequence will be replaced with single "." character if the sequence contains only "." characters, otherwise the sequence will be replaced with a single "*" character.

A resource database never contains more than one entry for a given ResourceName. If a resource file contains multiple lines with the same ResourceName, the last line in the file is used.

Any whitespace character before or after the name or colon in a ResourceSpec are ignored. To allow a Value to begin with whitespace, the two-character sequence ‘‘*space*’’ (backslash followed by space) is recognized and replaced by a space character, and the two-character sequence ‘‘*tab*’’ (backslash followed by horizontal tab) is recognized and replaced by a horizontal tab character. To allow a Value to contain embedded newline characters, the two-character sequence ‘‘\n’’ is recognized and replaced by a newline character. To allow a Value to be broken across multiple lines in a text file, the two-character sequence ‘‘*newline*’’ (backslash followed by newline) is recognized and removed from the value. To allow a Value to contain arbitrary character codes, the four-character sequence ‘‘*nnn*’’, where each *n* is a digit character in the range of ‘‘0’’–‘‘7’’, is recognized and replaced with a single byte that contains the octal value specified by the sequence. Finally, the two-character sequence ‘‘\\’’ is recognized and replaced with a single backslash.

When an application looks for the value of a resource, it specifies a complete path in the hierarchy, with both class and instance names. However, resource values are usually given with only partially specified names and classes, using pattern matching constructs. An asterisk (*) is a loose binding and is used to represent any number of intervening components, including none. A period (.) is a tight binding and is used to separate immediately adjacent components. A question mark (?) is used to match any single component name or class. A database entry cannot end in a loose binding; the final component (which cannot be "?") must be specified. The lookup algorithm searches the resource database for the entry that most closely matches (is most specific for) the full name and class being queried. When more than one database entry matches the full name and class, precedence rules are used to select just one.

The full name and class are scanned from left to right (from highest level in the hierarchy to lowest), one component at a time. At each level, the corresponding component and / or binding of each matching entry is determined, and these matching components and bindings are compared according to precedence rules. Each of the rules is applied at each level, before moving to the next level, until a rule selects a single entry over all others. The rules (in order of precedence) are:

1. An entry that contains a matching component (whether name, class, or "?") takes precedence over entries that elide the level (that is, entries that match the level in a loose binding).
2. An entry with a matching name takes precedence over both entries with a matching class and entries that match using "?". An entry with a matching class takes precedence over entries that match using "?".
3. An entry preceded by a tight binding takes precedence over entries preceded by a loose binding.

Programs based on the X Tookit Intrinsics obtain resources from the following sources (other programs usually support some subset of these sources):

RESOURCE_MANAGER root window property
: Any global resources that should be available to clients on all machines should be stored in the RESOURCE_MANAGER property on the root window of the first screen using the *xrdb* program. This is frequently taken care of when the user starts up X through the display manager or *xinit*.

SCREEN_RESOURCES root window property
: Any resources specific to a given screen (e.g. colors) that should be available to clients on all machines should be stored in the SCREEN_RESOURCES property on the root window of that screen. The *xrdb* program will sort resources automatically and place them in RESOURCE_MANAGER or SCREEN_RESOURCES, as appropriate.

application-specific files
: Directories named by the environment variable *XUSERFILESEARCHPATH* or the environment variable *XAPPLRESDIR*, plus directories in a standard place (usually under **$OPENNT_ROOT/usr/lib/X11/**, but this can be overridden with the *XFILESEARCHPATH* environment variable) are searched for for application-specific resources. For example, application default

resources are usually kept in **$OPENNT_ROOT/usr/lib/X11/app-defaults/**. See the *X Toolkit Intrinsics — C Language Interface* manual for details.

XENVIRONMENT

Any user- and machine-specific resources may be specified by setting the *XENVIRONMENT* environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, a file named **$HOME/.Xdefaults-***hostname* is looked for instead, where *hostname* is the name of the host where the application is executing.

–xrm *resourcestring*

Resources can also be specified from the command line. The *resourcestring* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of **–xrm** arguments may be given on the command line.

Program resources are organized into groups called *classes*, so that collections of individual resources (each of which are called *instances*) can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an upper case letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit Intrinsics will have at least the following resources:

background (class **Background)**

This resource specifies the color to use for the window background.

borderWidth (class **BorderWidth)**

This resource specifies the width in pixels of the window border.

borderColor (class **BorderColor)**

This resource specifies the color to use for the window border.

Most applications using the X Toolkit Intrinsics also have the resource **foreground** (class **Foreground**), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set Background and Foreground classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources. For example,

```
bitmap*Dashed:  off
XTerm*cursorColor:  gold
XTerm*multiScroll:  on
XTerm*jumpScroll:  on
XTerm*reverseWrap:  on
XTerm*curses:  on
XTerm*Font:  6x10
XTerm*scrollBar: on
XTerm*scrollbar*thickness: 5
```

```
XTerm*multiClickTime: 500
XTerm*charClass:  33:48,37:48,45-47:48,64:48
XTerm*cutNewline: off
XTerm*cutToBeginningOfLine: off
XTerm*titeInhibit:  on
XTerm*ttyModes:  intr ^c erase ^? kill ^u
XLoad*Background: gold
XLoad*Foreground: red
XLoad*highlight: black
XLoad*borderWidth: 0
emacs*Geometry:  80x65-0-0
emacs*Background:  rgb:5b/76/86
emacs*Foreground:  white
emacs*Cursor:  white
emacs*BorderColor:  white
emacs*Font:  6x10
xmag*geometry: -0-0
xmag*borderColor:  white
```

If these resources were stored in a file called **.Xresources** in your home directory, they could be added to any existing resources in the server with the following command:

```
% xrdb -merge $HOME/.Xresources
```

This is frequently how user-friendly startup scripts merge user-specific defaults into any site-wide defaults. All sites are encouraged to set up convenient ways of automatically loading resources. See the *Xlib* manual section *Resource Manager Functions* for more information.

EXAMPLES

The following is a collection of sample command lines for some of the more frequently used commands. For more information on a particular command, please refer to that command's manual page.

Load the resources file **$HOME/.Xresources:**

```
%  xrdb $HOME/.Xresources
```

Make the backspace key behave as the delete key:

```
%  xmodmap -e "keysym BackSpace = Delete"
```

Index the font directory **$OPENNT_ROOT/usr/local/lib/X11/otherfonts**:

```
%  mkfontdir $OPENNT_ROOT/usr/local/lib/X11/otherfonts
```

Add the font directory **$OPENNT_ROOT/usr/local/lib/X11/otherfonts** to the font path:

```
%  xset fp+ $OPENNT_ROOT/usr/local/lib/X11/otherfonts
```

Load the keymap file **$HOME/.keymap.km**:

```
%  xmodmap $HOME/.keymap.km
```

Set the root window to a light gray color:

```
% xsetroot -solid 'rgbi:.8/.8/.8'
```

Set user preferences: set the terminal bell to 100% volume at a pitch of 400 hertz, the volume of the keyclick noise to 50%, activate the screen saver after 1800 seconds, and turn key autorepeat on:

```
% xset b 100 400 c 50 s 1800 r on
```

Display the current user settings:

```
% xset q
```

Start the *twm* window manager:

```
% twm
```

Start the *xmag* program (to magnify portions of the screen):

```
% xmag
```

Start the clock program with a size of 48 pixels square in the upper right-hand corner, with a blue background and a white foreground:

```
% xclock -geometry 48x48-0+0 -bg blue -fg white
```

Start the *xeyes* program with a size of 48 pixels square, 48 pixels from the upper right corner (beside the *xclock* of the previous example):

```
% xeyes -geometry 48x48-48+0
```

Start the *xbiff* program to check for mail, and have it check for mail every 20 seconds:

```
% xbiff -update 20
```

List fonts that match the pattern *helvetica*:

```
% xlsfonts '*helvetica*'
```

Display information about the root window:

```
% xwininfo -root
```

Display X information about the display named *joesworkstation:0*:

```
% xdpyinfo -display joesworkstation:0
```

Remove the host *joesworkstation:0* from the list of allowable hosts:

```
% xhost -joesworkstation
```

Update the screen:

```
% xrefresh
```

Capture an image of an X window and display it:

```
% xwd | xwud
```

Create or edit the bitmap **companylogo.bm** which is 32 pixels square:

```
% bitmap companylogo.bm 32x32
```

Start the *xcalc* (calculator) program with a blue background and a magenta foreground:

```
% xcalc -bg blue -fg magenta
```

Start a new *xterm* session 80 columns wide and 66 rows high in the lower right corner with the name **myexterm** and any other arguments passed to the command:

```
% xterm -geometry 80x66-0-0 -name myxterm $*
```

Use *xon* to start the *xload* program remotely, on the machine **filesysmachine**:

```
% xon filesysmachine xload
```

DIAGNOSTICS

A wide variety of error messages are generated from various programs. The default error handler in *Xlib* (also used by many toolkits) uses standard resources to construct diagnostic messages when errors occur. The defaults for these messages are usually stored in **$OPENNT_ROOT/usr/lib/X11/XErrorDB**. If this file is not present, error messages will be rather terse and cryptic.

When the X Toolkit Intrinsics encounter errors converting resource strings to the appropriate internal format, no error messages are usually printed. This is convenient when it is desirable to have one set of resources across a variety of displays (e.g. color vs. monochrome, lots of fonts vs. very few, etc.), although it can pose problems for trying to determine why an application might be failing. This behavior can be overridden by the setting the *StringConversionsWarning* resource.

To force the X Toolkit Intrinsics to always print string conversion error messages, the following resource should be placed in the file that gets loaded onto the RESOURCE_MANAGER property using the *xrdb* program (frequently called **.Xresources** or **.Xres** in the user's home directory):

```
*StringConversionWarnings: on
```

To have conversion messages printed for just a particular application, the appropriate instance name can be placed before the asterisk:

```
xterm*StringConversionWarnings: on
```

SEE ALSO

Introductions:

XConsortium, *XStandards*, *XSecurity*

The following are clients, utilities, and demos:

appres, *bdftopcf*, *bitmap*, *editres*, *fs*, *fsinfo*, *fslsfonts*, *fstobdf*, *ico*, *imake*, *listres*, *lndir*, *makedepend*, *maze*, *mkdirhier*, *mkfontdir*, *oclock*, *puzzle*, *showfont*, *twm*, *viewres*, *x11perf*, *x11perfcomp*, *xauth*, *xbiff*, *xcalc*, *xclipboard*, *xclock*, *xcmsdb*, *xcmstest*, *xconsole*, *xcutsel*, *xditview*, *xdpr*, *xdpyinfo*, *xedit*, *xev*, *xeyes*, *xfd*, *xfontsel*, *xgc*, *xkill*, *xlogo*, *xlsatoms*, *xlsclients*, *xlsfonts*, *xmag*, *xman*, *xmkmf*, *xmodmap*, *xon*, *xpr*, *xprop*, *xrdb*, *xrefresh*, *xset*, *xsetroot*, *xstdcmap*, *xwd*, *xwininfo*, *xwud*

The following are servers:

Xserver, *Xdec*, *XmacII*, *Xmips*, *Xqdss*, *Xqvss*, *Xsun*, *X386*, *kbd_mode*

These specifications may also be of interest:

Xlib – C Language X Interface, and *X Toolkit Intrinsics – C Language Interface*

COPYRIGHT

The following copyright and permission notice outlines the rights and restrictions covering most parts of the core distribution of the X Window System from MIT. Other parts have additional or different copyrights and permissions; see the individual source files.

Copyright 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991 by the Massachusetts Institute of Technology.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. MIT makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

TRADEMARKS

X Window System is a trademark of MIT.

AUTHORS

A cast of thousands, literally. The MIT Release 5 distribution is brought to you by the MIT X Consortium. The names of all people who made it a reality will be found in the individual documents and source files. The staff members at MIT responsible for this release are: Donna Converse (MIT X Consortium), Stephen Gildea (MIT X Consortium), Susan Hardy (MIT X Consortium), Jay Hersh (MIT X Consortium), Keith Packard (MIT X Consortium), David Sternlicht (MIT X Consortium), Bob Scheifler (MIT X Consortium), and Ralph Swick (Digital / MIT Project Athena).

NAME

MIT X Consortium – MIT X Consortium information

SYNOPSIS

Release 5 of X Version 11 is brought to you by the MIT X Consortium.

DESCRIPTION

X Window System research began in the summer of 1984 at MIT as an informal joint undertaking between the Laboratory for Computer Science and Project Athena. Since that time, researchers and engineers at numerous other universities and companies have been involved in the design and implementation.

The MIT X Consortium was formed in January of 1988 to further the development of the X Window System. It is part of the Laboratory for Computer Science at MIT, and has as its major goal the promotion of cooperation within the computer industry in the creation of standard software interfaces at all layers in the X Window System environment. MIT's role is to provide the vendor-neutral architectural and administrative leadership required to make this work. The Consortium is financially self-supporting, with membership open to any organization. There are two categories of membership, Member (for large organizations) and Affiliate (for smaller organizations). A list of members as of the release of X11R5 is given below.

The Director of the X Consortium acts as the ultimate architect for all X specifications and software. The activities of the Consortium are overseen by an MIT Steering Committee, which helps set policy and provides strategic guidance and review of the Consortium's activities. An Advisory Committee made up of member representatives meets regularly to review the Consortium's plans, assess its progress, and suggest future directions.

Most of the Consortium's activities take place via electronic mail, with meetings when required. As designs and specifications take shape, interest groups are formed from experts in the participating organizations. Typically a small multi-organization architecture team leads the design, with others acting as close observers and reviewers. Once a complete specification is produced, it may be submitted for formal technical review by the Consortium as a proposed standard. The standards process includes public review (outside the Consortium) and a demonstration of proof of concept, typically established with a public, portable implementation of the specification, (although other methods are possible on a case by case basis).

The MIT staff of the Consortium also maintains a software and documentation collection, containing both Consortium-developed and user-contributed materials, and endeavors to make periodic distributions of this collection available to the public without license and for a minimal fee. X11R5 is the fifth such distribution. The Consortium also sponsors an annual conference, open to the public, to promote the exchange of technical information about X.

STAFF

The Director of the X Consortium is Bob Scheifler, rws@expo.lcs.mit.edu. The staff of the X Consortium are all full-time research staff employed by the MIT Laboratory for Computer Science. The staff members are as follows.

Donna Converse, converse@expo.lcs.mit.edu
Stephen Gildea, gildea@expo.lcs.mit.edu
Susan Hardy, sh@expo.lcs.mit.edu
Jay Hersh, hersh@expo.lcs.mit.edu
Keith Packard, keith@expo.lcs.mit.edu
Dave Sternlicht, dave@expo.lcs.mit.edu

Ralph Swick, an employee of Digital Equipment Corporation working at MIT Project Athena, also worked directly with the Consortium staff on this release. Former MIT staff members Jim Fulton and Chris Peterson contributed to the early development of the release.

POSTAL ADDRESS

The MIT X Consortium can be reached by writing to

Bob Scheifler
MIT X Consortium
Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

MEMBERS

Apple Computer, Inc.
Bull S.A.
Control Data Corporation
Convex Computer Corporation
Cray Research, Inc.
Data General Corporation
Digital Equipment Corp.
E.I. DuPont de Nemours & Company, Inc.
Eastman Kodak Company
Evans & Sutherland
Fujitsu Limited
Hewlett-Packard Company
Hitachi Ltd.
Hughes Aircraft Company
IBM Corporation
Intergraph Corp.
Kubota Corp.
Matsushita Electric Industrical Co., Ltd.
MIPS Computer Systems
Mitsubishi Electric Corporation
Motorola, Inc.
NCR Corporation
NEC Corporation
Network Computing Devices
Nippon Telegraph and Telephone Corporation
Oki Electric Industry Co., Ltd.
Olivetti Systems & Networks
OMRON Corporation
Prime Computer, Inc.
The Santa Cruz Operation, Inc.
Sequent Computer Systems Inc.

Siemens Nixdorf Informationssysteme AG
Silicon Graphics, Inc.
Sony Corporation
Sun Microsystems, Inc.
Tandberg Data A/S
Tektronix, Inc.
Texas Instruments, Inc.
Unisys Corporation
Unix Systems Laboratories/American Telephone & Telegraph Company
Xerox Corporation

AFFILIATES

Adobe Systems Inc.
AGE Logic, Inc.
ASTEC, Inc.
Athenix Corporation
Bitstream, Inc.
CETIA - Compagnie Europeene des Techniques de l'Ingenierie Assistee
Chromatics, Inc.
Codonics, Inc.
Computer & Communication Research Laboratories
Frame Technology Corp.
Gfxbase, Inc.
GIPSI S.A.
HaL Computer Systems, Inc.
Institute for Information Industry
Interactive Systems Corporation
Integrated Computer Solutions, Inc.
IXI Limited
Japan Computer Corporation
Jupiter Systems
KAIST - Korean Advanced Institute of Science and Technology
Labtam Australia
Locus Computing Corporation
Metro Link, Inc.
University of Lowell
Megatek Corporation
Metheus Corporation
MITRE Corporation
Objectivity, Inc.
Open Software Foundation
O'Reilly & Associates, Inc.
PCS Computer Systeme GmbH
Peritek Corp.
PsiTech, Inc.
Quarterdeck Office Systems
Ramtek Corporation
Samsung Electronics Corp.
ShoGraphics, Inc.
Snitily Graphics Consulting Services
Solbourne Computer, Inc.
SOUM Corporation

SPARC International
Spectragraphics Corp.
Stanford University
Stardent Computer
Strategic Research Institute Inc.
Sumitomo Electric Workstation
Tatung Science and Technology
Template Graphics Software
Tyan Computer
Unipalm XTech
Visix Software, Inc.
Visual Information Technologies, Inc.
Visual Technology, Inc.
X/Open Company Ltd.

NAME

X Security – X display access control

SYNOPSIS

X provides mechanism for implementing many access control systems. Release 5 includes four mechanisms:

Host Access	Simple host-based access control.
MIT-MAGIC-COOKIE-1	Shared plain-text "cookies".
XDM-AUTHORIZATION-1	Secure DES based private-keys.
SUN-DES-1	Based on Sun's secure rpc system.

ACCESS SYSTEM DESCRIPTIONS

Host Access

Any client on a host in the host access control list is allowed access to the X server. This system can work reasonably well in an environment where everyone trusts everyone, or when only a single person can log in to a given machine, and is easy to use when the list of hosts used is small. This system does not work well when multiple people can log in to a single machine and mutual trust does not exist. The list of allowed hosts is stored in the X server and can be changed with the *xhost* command. When using the more secure mechanisms listed below, the host list is normally configured to be the empty list, so that only authorized programs can connect to the display.

MIT-MAGIC-COOKIE-1

When using MIT-MAGIC-COOKIE-1, the client sends a 128 bit "cookie" along with the connection setup information. If the cookie presented by the client matches one that the X server has, the connection is allowed access. The cookie is chosen so that it is hard to guess; *xdm* generates such cookies automatically when this form of access control is used. The user's copy of the cookie is usually stored in the **.Xauthority** file in the home directory, although the environment variable *XAUTHORITY* can be used to specify an alternate location. *Xdm* automatically passes a cookie to the server for each new login session, and stores the cookie in the user file at login.

The cookie is transmitted on the network without encryption, so there is nothing to prevent a network snooper from obtaining the data and using it to gain access to the X server. This system is useful in an environment where many users are running applications on the same machine and want to avoid interference from each other, with the caveat that this control is only as good as the access control to the physical network. In environments where network-level snooping is difficult, this system can work reasonably well.

XDM-AUTHORIZATION-1

For sites in the US, Release 5 contains a DES-based access control mechanism called XDM-AUTHORIZATION-1. It is similar in usage to MIT-MAGIC-COOKIE-1 in that a key is stored in the **.Xauthority** file and is shared with the X server. However, this key consists of two parts - a 56 bit DES encryption key and 64 bits of random data used as

the authenticator.

When connecting to the X server, the application generates 192 bits of data by combining the current time in seconds (since 00:00 1/1/1970 GMT) along with 48 bits of "identifier". For TCP/IP connections, the identifier is the address plus port number; for local connections it is the process ID and 32 bits to form a unique id (in case multiple connections to the same server are made from a single process). This 192 bit packet is then encrypted using the DES key and sent to the X server, which is able to verify if the requestor is authorized to connect by decrypting with the same DES key and validating the authenticator and additional data. This system is useful in many environments where host-based access control is inappropriate and where network security cannot be ensured.

SUN-DES-1

Recent versions of SunOS (and some other systems) have included a secure public key remote procedure call system. This system is based on the notion of a network principal; a user name and NIS domain pair. Using this system, the X server can securely discover the actual user name of the requesting process. It involves encrypting data with the X servers public key, and so the identity of the user who started the X server is needed for this; this identity is stored in the **.Xauthority** file. By extending the semantics of "host address" to include this notion of network principal, this form of access control is very easy to use. To allow access by a new user, use *xhost*. For example,

```
xhost keith@ joe@mit.edu
```

adds "keith" from the NIS domain of the local machine, and "joe" in the "mit.edu" NIS domain. For keith or joe to successfully connect to the display, they must add the principal who started the server to their **.Xauthority** file. For example (line-break inserted for readability):

```
xauth add expo.lcs.mit.edu:0 SUN-DES-1 \
       xinu.expo.lcs.mit.edu:x.lcs.mit.edu
```

This system only works on machines which support Secure RPC, and only for users which have set up the appropriate public/private key pairs on their system. See the Secure RPC documentation for details.

THE AUTHORIZATION FILE

Except for Host Access control, each of these systems uses data stored in the **.Xauthority** file to generate the correct authorization information to pass along to the X server at connection setup. MIT-MAGIC-COOKIE-1 and XDM-AUTHORIZATION-1 store secret data in the file; so anyone who can read the file can gain access to the X server. SUN-DES-1 stores only the identity of the principal who started the server (xinu.*hostname@domain* when the server is started by *xdm*), and so it is not useful to anyone not authorized to connect to the server.

Each entry in the **.Xauthority** file matches a certain connection family (TCP/IP, DECnet or local connections) and X display name (hostname plus display number). This allows multiple authorization entries for different displays to

share the same data file. A special connection family (FamilyWild, value 65535) causes an entry to match every display, allowing the entry to be used for all connections. Each entry additionally contains the authorization name and whatever private authorization data is needed by that authorization type to generate the correct information at connection setup time.

The *xauth* program manipulates the **.Xauthority** file format. It understands the semantics of the connection families and address formats, displaying them in an easy to understand format. It also understands that SUN-DES-1 uses string values for the authorization data, and displays them appropriately.

The X server (when running on a workstation) reads authorization information from a file name passed on the command line with the **–auth** option (see the *Xserver* manual page). The authorization entries in the file are used to control access to the server. In each of the authorization schemes listed above, the data needed by the server to initialize an authorization scheme is identical to the data needed by the client to generate the appropriate authorization information, so the same file can be used by both processes. This is especially useful when *xinit* is used.

MIT-MAGIC-COOKIE-1
: This system uses 128 bits of data shared between the user and the X server. Any collection of bits can be used. *Xdm* generates these keys using a cryptographically secure pseudo random number generator, and so the key to the next session cannot be computed from the current session key.

XDM-AUTHORIZATION-1
: This system uses two pieces of information. First, 64 bits of random data, second a 56 bit DES encryption key (again, random data) stored in 8 bytes, the last byte of which is ignored. *Xdm* generates these keys using the same random number generator as is used for MIT-MAGIC-COOKIE-1.

SUN-DES-1
: This system needs a string representation of the principal which identifies the associated X server. When *xdm* starts the X server, it uses the root principal for the machine on which it is running (xinu.*hostname@domain*, e.g. "xinu.expire.lcs.mit.edu@x.lcs.mit.edu"). Putting the correct principal name in the **.Xauthority** file causes Xlib to generate the appropriate authorization information using the secure RPC library.

FILES

.Xauthority

SEE ALSO

X, xdm, xauth, xhost, xinit, Xserver

NAME

X Standards – MIT X Consortium Standards

SYNOPSIS

The major goal of the MIT X Consortium is to promote cooperation within the computer industry in the creation of standard software interfaces at all layers in the X Window System environment. The status of various standards and the software in the X11R5 distribution is explained below.

STANDARDS

The following documents are MIT X Consortium standards:

- *X Window System Protocol*, X Version 11, Release 5, Robert W. Scheifler
- *Xlib — C Language X Interface*, X Version 11, Release 5, James Gettys, Robert W. Scheifler, Ron Newman
- *X Toolkit Intrinsics — C Language Interface*, X Version 11, Release 5 Joel McCormack, Paul Asente, Ralph R. Swick
- *Bitmap Distribution Format*, Version 2.1, X Version 11, Release 5
- *Inter-Client Communication Conventions Manual*, Version 1.1, X Version 11, Release 5, David Rosenthal
- *Compound Text Encoding*, Version 1.1, X Version 11, Release 5, Robert W. Scheifler
- *X Logical Font Description Conventions*, Version 1.4, X Version 11, Release 5, Jim Flowers
- *X Display Manager Control Protocol*, Version 1.0, X Version 11, Release 5, Keith Packard
- *X11 Nonrectangular Window Shape Extension*, Version 1.0, X Version 11, Release 5, Keith Packard
- *X11 Input Extension Protocol Specification*, Version 1.0, X Version 11, Release 5, George Sachs, Mark Patrick
- *X11 Input Extension Library Specification*, X Version 11, Release 5, Mark Patrick, George Sachs
- *The X Font Service Protocol*, Version 1.0, X Version 11, Release 5, Jim Fulton

DRAFT STANDARDS

The following documents are currently draft standards of the MIT X Consortium.

- *PEX Protocol Specification*, Version 5.0P, Sally Barry (editor)

  Extending X for Double-Buffering, Multi-Buffering, and Stereo, Version 3.2, Jeffrey Friedberg, Larry Seiler, Jeff Vroom

INCLUDE FILES

The following include files are part of the Xlib standard.

```
<X11/cursorfont.h>
<X11/keysym.h>
```

```
<X11/keysymdef.h>
<X11/X.h>
<X11/Xatom.h>
<X11/Xcms.h>
<X11/Xlib.h>
<X11/Xlibint.h>
<X11/Xproto.h>
<X11/Xprotostr.h>
<X11/Xresource.h>
<X11/Xutil.h>
<X11/X10.h>
```

The following include files are part of the X Toolkit Intrinsics standard.

```
<X11/Composite.h>
<X11/CompositeP.h>
<X11/Constraint.h>
<X11/ConstrainP.h>
<X11/Core.h>
<X11/CoreP.h>
<X11/Intrinsic.h>
<X11/IntrinsicP.h>
<X11/Object.h>
<X11/ObjectP.h>
<X11/RectObj.h>
<X11/RectObjP.h>
<X11/Shell.h>
<X11/ShellP.h>
<X11/StringDefs.h>
<X11/Vendor.h>
<X11/VendorP.h>
```

The following include file is part of the Nonrectangular Window Shape Extension standard.

```
<X11/extensions/shape.h>
```

The following include files are part of the X Input Extension standard.

```
<X11/extensions/XI.h>
<X11/extensions/XInput.h>
<X11/extensions/XIproto.h>
```

The following include file is part of the Multi-Buffering draft standard.

```
<X11/extensions/multibuf.h>
```

NON STANDARDS

The X11R5 distribution contains *sample* implementations, not *reference* implementations. Although much of the code is believed to be correct, the code should be assumed to be in error wherever it conflicts with the specification.

At the public release of X11R5, the only MIT X Consortium standards are the ones listed above. No other documents, include files, or software in X11R5 carry special status within the X Consortium. For example, none of the

following are standards: internal interfaces of the sample server; the MIT-SHM extension; the Input Synthesis extension; the Athena Widget Set; the Xmu library; the Xau library; CLX, the Common Lisp interface to X; the RGB database; the fonts distributed with X11R5; the applications distributed with X11R5; the include files **<X11/XWDFile.h>**, **<X11/Xfuncproto.h>**, **<X11/Xfuncs.h,> <X11/Xosdefs.h>**, and **<X11/Xos.h>**; the bitmap files in **<X11/bitmaps>**.

NAME

x11perf – X11 server performance test program

SYNOPSIS

`x11perf [-`*`option ...`* `]`

DESCRIPTION

The *x11perf* program runs one or more performance tests and reports how fast an X server can execute the tests.

Many graphics benchmarks assume that the graphics device is used to display the output of a single fancy graphics application, and that the user gets his work done on some other device, like a terminal. Such benchmarks usually measure drawing speed for lines, polygons, text, etc.

Since workstations are not used as standalone graphics engines, but as super-terminals, *x11perf* measures window management performance as well as traditional graphics performace. *x11perf* includes benchmarks for the time it takes to create and map windows (as when you start up an application); to map a pre-existing set of windows onto the screen (as when you deiconify an application or pop up a menu); and to rearrange windows (as when you slosh windows to and fro trying to find the one you want).

x11perf also measures graphics performance for operations not normally used in standalone graphics displays, but are nonetheless used frequently by X applications. Such operations include CopyPlane (used to map bitmaps into pixels), scrolling (used in text windows), and various stipples and tiles (used for CAD and color half-toning, respectively).

x11perf should be used to analyze particular strengths and weaknesses of servers, and is most useful to a server writer who wants to analyze and improve a server. *x11perf* is meant to comprehensively exercise just about every X11 operation you can perform; it does not purport to be a representative sample of the operations that X11 applications actually use. While it can be used as a benchmark, it was written and is intended as a performance testing tool.

As such, *x11perf* **does not** whittle down measurements to a single ''HeXStones'' or ''MeXops'' number. We consider such numbers to be uninformative at best and misleading at worst. Some servers which are very fast for certain applications can be very slow for others. No single number or small set of numbers are sufficient to characterize how an X implementation will perform on all applications. However, by knowledge of your favorite application, you may be able to use the numbers *x11perf* reports to predict its performance on a given X implementation.

That said, you might also want to look at *x11perfcomp*, a program to compare the outputs of different *x11perf* runs. You provide a list of files containing results from *x11perf*, and it lays them out in a nice tabular format.

For repeatable results, *x11perf* should be run using a local connection on a freshly-started server. The default configuration runs each test 5 times, in order to see if each trial takes approximately the same amount of time. Strange glitches should be examined; if non-repeatable one might chalk them up to

daemons and network traffic. Each trial is run for 5 seconds, in order to reduce random time differences. The number of objects processed per second is displayed to 3 significant digits, but you'll be lucky on most systems if the numbers are actually consistent to 2 digits. *x11perf* moves the cursor out of the test window; you should be careful not to bump the mouse and move it back into the window. (A prize to people who correctly explain why!!).

Before running a test, *x11perf* determines what the round trip time to the server is, and factors this out of the final timing reported. It ensures that the server has actually performed the work requested by fetching a pixel back from the test window, which means that servers talking to graphics accelerators can't claim that they are done, while in the meantime the accelerator is painting madly.

By default *x11perf* automatically calibrates the number of repetitions of each test, so that each should take approximately the same length of time to run across servers of widely differing speeds. However, since each test must be run to completion at least once, some slow servers may take a very long time, particularly on the window moving and resizing tests, and on the arc drawing tests.

All timing reports are for the smallest object involved. For example, the line tests use a PolyLine request to paint several lines at once, but report how many lines per second the server can paint, not how many PolyLine requests per second. Text tests paint a line of characters, but report on the number of characters per second. Some window tests map, unmap, or move a single parent window, but report on how many children windows per second the server can map, unmap, or move.

The current program is mostly the responsibility of Joel McCormack. It is based upon the x11perf developed by Phil Karlton, Susan Angebranndt, and Chris Kent, who wanted to assess performance differences between various servers. Mary Walker, Todd Newman, and several tests were added in order to write and tune the PMAX (DECStation 3100) servers. For a general release to the world, *x11perf* was rewritten to ease making comparisons between widely varying machines, to cover most important (and unimportant) X functionality, and to exercise graphics operations in as many different orientations and alignments as possible.

OPTIONS

x11perf is solely Xlib based, and accepts the options listed below:

–display *host:dpy*
Specifies which display to use.

–sync Runs the tests in synchronous mode. Normally only useful for debugging *x11perf*.

–pack Runs rectangle tests so that they pack rectangles right next to each other. This makes it easy to debug server code for stipples and tiles...if the pattern looks ugly, you've got alignment problems.

–repeat <*n*>
Repeats each test *n* times (by default each test is run 5 times).

–time *<s>*
Specifies how long in seconds each test should be run (default 5 seconds).

–all Runs all tests. This may take a while.

–range *<test1>[,<test2>]*
Runs all the tests starting from the specified name *test1* until the name *test2*, including both the specified tests. The testnames should be one of the options starting from -dot. (eg) -range line100 will peform the tests from the 100 pixel line test, and go on till the last test, -range line100,dline10 will do the tests from line100 to dline10.

–labels Generates just the descriptive labels for each test specified. See *x11perfcomp* for more details.

–fg *color*
Specifies the foreground color to use.

–bg *color*
Specifies the background color to use.

–rop *<rop0 rop1 ...>*
Use specified raster ops (default is GXcopy). This option only affects graphics benchmarks in which the graphics function is actually used.

–pm *<pm0 pm1 ...>*
Use specified planemasks (default is ~0). This option only affects graphics benchmarks in which the planemask is actually used.

–depth *<depth>*
Use a visual with <depth> planes per pixel (default is the default visual).

–dot Dot.

–rect1 1x1 solid-filled rectangle.

–rect10 10x10 solid-filled rectangle.

–rect100 100x100 solid-filled rectangle.

–rect500
500x500 solid-filled rectangle.

–srect1 1x1 transparent stippled rectangle, 8x8 stipple pattern.

–srect10 10x10 transparent stippled rectangle, 8x8 stipple pattern.

–srect100
100x100 transparent stippled rectangle, 8x8 stipple pattern.

–srect500
500x500 transparent stippled rectangle, 8x8 stipple pattern.

–osrect1 1x1 opaque stippled rectangle, 8x8 stipple pattern.

–osrect10
10x10 opaque stippled rectangle, 8x8 stipple pattern.

–osrect100
100x100 opaque stippled rectangle, 8x8 stipple pattern.

–osrect500
500x500 opaque stippled rectangle, 8x8 stipple pattern.

–tilerect1
1x1 tiled rectangle, 4x4 tile pattern.

–tilerect10
10x10 tiled rectangle, 4x4 tile pattern.

–tilerect100
100x100 tiled rectangle, 4x4 tile pattern.

–tilerect500
500x500 tiled rectangle, 4x4 tile pattern.

–bigsrect1
1x1 stippled rectangle, 161x145 stipple pattern.

–bigsrect10
10x10 stippled rectangle, 161x145 stipple pattern.

–bigsrect100
100x100 stippled rectangle, 161x145 stipple pattern.

–bigsrect500
500x500 stippled rectangle, 161x145 stipple pattern.

–bigosrect1
1x1 opaque stippled rectangle, 161x145 stipple pattern.

–bigosrect10
10x10 opaque stippled rectangle, 161x145 stipple pattern.

–bigosrect100
100x100 opaque stippled rectangle, 161x145 stipple pattern.

–bigosrect500
500x500 opaque stippled rectangle, 161x145 stipple pattern.

–bigtilerect1
1x1 tiled rectangle, 161x145 tile pattern.

–bigtilerect10
10x10 tiled rectangle, 161x145 tile pattern.

–bigtilerect100
100x100 tiled rectangle, 161x145 tile pattern.

–bigtilerect500
500x500 tiled rectangle, 161x145 tile pattern.

–eschertilerect1
1x1 tiled rectangle, 215x208 tile pattern.

–eschertilerect10
10x10 tiled rectangle, 215x208 tile pattern.

–eschertilerect100
100x100 tiled rectangle, 215x208 tile pattern.

–eschertilerect500
500x500 tiled rectangle, 215x208 tile pattern.

–seg1 1-pixel thin line segment.

–seg10 10-pixel thin line segment.

–seg100 100-pixel thin line segment.

–seg500
500-pixel thin line segment.

–seg100c1
100-pixel thin line segment (1 obscuring rectangle).

–seg100c2
100-pixel thin line segment (2 obscuring rectangles).

–seg100c3
100-pixel thin line segment (3 obscuring rectangles).

–dseg10 10-pixel thin dashed segment (3 on, 2 off).

–dseg100
100-pixel thin dashed segment (3 on, 2 off).

–ddseg100
100-pixel thin double-dashed segment (3 fg, 2 bg).

–hseg10
10-pixel thin horizontal line segment.

–hseg100
100-pixel thin horizontal line segment.

–hseg500
500-pixel thin horizontal line segment.

–vseg10 10-pixel thin vertical line segment.

–vseg100
100-pixel thin vertical line segment.

–vseg500
500-pixel thin vertical line segment.

–whseg10
10-pixel wide horizontal line segment.

–whseg100
100-pixel wide horizontal line segment.

–whseg500
500-pixel wide horizontal line segment.

–wvseg10
10-pixel wide vertical line segment.

–wvseg100
100-pixel wide vertical line segment.

–wvseg500
500-pixel wide vertical line segment.

–line1 1-pixel thin (width 0) line.

–line10 10-pixel thin line.

–line100
100-pixel thin line.

–line500
500-pixel thin line.

–dline10
10-pixel thin dashed line (3 on, 2 off).

–dline100
100-pixel thin dashed line (3 on, 2 off).

–ddline100
100-pixel thin double-dashed line (3 fg, 2 bg).

–wline10
10-pixel line, line width 1.

–wline100
100-pixel line, line width 10.

–wline500
500-pixel line, line width 50.

–wdline100
100-pixel dashed line, line width 10 (30 on, 20 off).

–wddline100
100-pixel double-dashed line, line width 10 (30 fg, 20 bg).

–orect10 10x10 thin rectangle outline.

–orect100
100-pixel thin vertical line segment.

–orect500
500-pixel thin vertical line segment.

–worect10
10x10 wide rectangle outline.

–worect100
100-pixel wide vertical line segment.

–worect500
500-pixel wide vertical line segment.

–circle1 1-pixel diameter thin (line width 0) circle.

–circle10
10-pixel diameter thin circle.

–circle100
100-pixel diameter thin circle.

–circle500
500-pixel diameter thin circle.

–dcircle100
100-pixel diameter thin dashed circle (3 on, 2 off).

–ddcircle100
100-pixel diameter thin double-dashed circle (3 fg, 2 bg).

–wcircle10
10-pixel diameter circle, line width 1.

–wcircle100
100-pixel diameter circle, line width 10.

–wcircle500
500-pixel diameter circle, line width 50.

–wdcircle100
100-pixel diameter dashed circle, line width 10 (30 on, 20 off).

–wddcircle100
100-pixel diameter double-dashed circle, line width 10 (30 fg, 20 bg).

–pcircle10
10-pixel diameter thin partial circle, orientation and arc angle evenly distributed.

–pcircle100
100-pixel diameter thin partial circle.

–wpcircle10
10-pixel diameter wide partial circle.

–wpcircle100
100-pixel diameter wide partial circle.

–fcircle1
1-pixel diameter filled circle.

–fcircle10
10-pixel diameter filled circle.

–fcircle100
100-pixel diameter filled circle.

–fcircle500
500-pixel diameter filled circle.

–fcpcircle10
10-pixel diameter partial filled circle, chord fill, orientation and arc angle evenly distributed.

–fcpcircle100
100-pixel diameter partial filled circle, chord fill.

–fspcircle10
10-pixel diameter partial filled circle, pie slice fill, orientation and arc angle evenly distributed.

–fspcircle100
100-pixel diameter partial filled circle, pie slice fill.

–ellipse10
10-pixel diameter thin (line width 0) ellipse, major and minor axis sizes evenly distributed.

–ellipse100
100-pixel diameter thin ellipse.

–ellipse500
500-pixel diameter thin ellipse.

–dellipse100
100-pixel diameter thin dashed ellipse (3 on, 2 off).

–ddellipse100
100-pixel diameter thin double-dashed ellipse (3 fg, 2 bg).

–wellipse10
10-pixel diameter ellipse, line width 1.

–wellipse100
100-pixel diameter ellipse, line width 10.

–wellipse500
500-pixel diameter ellipse, line width 50.

–wdellipse100
100-pixel diameter dashed ellipse, line width 10 (30 on, 20 off).

–wddellipse100
100-pixel diameter double-dashed ellipse, line width 10 (30 fg, 20 bg).

–pellipse10
10-pixel diameter thin partial ellipse.

–pellipse100
100-pixel diameter thin partial ellipse.

–wpellipse10
10-pixel diameter wide partial ellipse.

–wpellipse100
100-pixel diameter wide partial ellipse.

–fellipse10
10-pixel diameter filled ellipse.

–fellipse100
100-pixel diameter filled ellipse.

–fellipse500
500-pixel diameter filled ellipse.

–fcpellipse10
10-pixel diameter partial filled ellipse, chord fill.

–fcpellipse100
100-pixel diameter partial filled ellipse, chord fill.

–fspellipse10
10-pixel diameter partial filled ellipse, pie slice fill.

–fspellipse100
100-pixel diameter partial filled ellipse, pie slice fill.

–triangle1
Fill 1-pixel / side triangle.

–triangle10
Fill 10-pixel / side triangle.

–triangle100
Fill 100-pixel / side triangle.

–trap10 Fill 10x10 trapezoid.

–trap100
Fill 100x100 trapezoid.

–strap10
Fill 10x10 transparent stippled trapezoid, 8x8 stipple pattern.

–strap100
Fill 100x100 transparent stippled trapezoid, 8x8 stipple pattern.

–ostrap10
Fill 10x10 opaque stippled trapezoid, 8x8 stipple pattern.

–ostrap100
Fill 100x100 opaque stippled trapezoid, 8x8 stipple pattern.

–tiletrap10
Fill 10x10 tiled trapezoid, 4x4 tile pattern.

–tiletrap100
Fill 100x100 tiled trapezoid, 4x4 tile pattern.

–bigstrap10
Fill 10x10 transparent stippled trapezoid, 161x145 stipple pattern.

–bigstrap100
Fill 100x100 transparent stippled trapezoid, 161x145 stipple pattern.

–bigostrap10
Fill 10x10 opaque stippled trapezoid, 161x145 stipple pattern.

–bigostrap100
Fill 100x100 opaque stippled trapezoid, 161x145 stipple pattern.

–bigtiletrap10
Fill 10x10 tiled trapezoid, 161x145 tile pattern.

–bigtiletrap100
Fill 100x100 tiled trapezoid, 161x145 tile pattern.

–eschertiletrap10
Fill 10x10 tiled trapezoid, 216x208 tile pattern.

–eschertiletrap100
Fill 100x100 tiled trapezoid, 216x208 tile pattern.

–complex10
Fill 10-pixel / side complex polygon.

–complex100
Fill 100-pixel / side complex polygon.

–ftext Character in 80-char line (6x13).

–f8text Character in 70-char line (8x13).

–f9text Character in 60-char line (9x15).

–f14text16
2-byte character in 40-char line (k14).

–tr10text
Character in 80-char line (Times-Roman 10).

–tr24text
Character in 30-char line (Times-Roman 24).

–polytext
Character in 20 / 40 / 20 line (6x13, Times-Roman 10, 6x13).

–fitext Character in 80-char image line (6x13).

–f8itext Character in 70-char image line (8x13).

–f9itext Character in 60-char image line (9x15).

–f14itext16
2-byte character in 40-char image line (k14).

–tr10itext
Character in 80-char image line (Times-Roman 10).

–tr24itext
Character in 30-char image line (Times-Roman 24).

–scroll10
Scroll 10x10 pixels vertically.

–scroll100
Scroll 100x100 pixels vertically.

–scroll500
Scroll 500x500 pixels vertically.

–copywinwin10
Copy 10x10 square from window to window.

–copywinwin100
Copy 100x100 square from window to window.

–copywinwin500
Copy 500x500 square from window to window.

–copypixwin10
Copy 10x10 square from pixmap to window.

–copypixwin100
Copy 100x100 square from pixmap to window.

–copypixwin500
Copy 500x500 square from pixmap to window.

–copywinpix10
Copy 10x10 square from window to pixmap.

–copywinpix100
Copy 100x100 square from window to pixmap.

–copywinpix500
Copy 500x500 square from window to pixmap.

–copypixpix10
Copy 10x10 square from pixmap to pixmap.

–copypixpix100
Copy 100x100 square from pixmap to pixmap.

–copypixpix500
Copy 500x500 square from pixmap to pixmap.

–copyplane10
Copy 10x10 1-bit deep plane.

–copyplane100
Copy 100x100 1-bit deep plane.

–copyplane500
Copy 500x500 1-bit deep plane.

–putimage10
PutImage 10x10 square.

–putimage100
PutImage 100x100 square.

–putimage500
PutImage 500x500 square.

–shmput10
PutImage 10x10 square, MIT shared memory extension.

–shmput100
PutImage 100x100 square, MIT shared memory extension.

–shmput500
PutImage 500x500 square, MIT shared memory extension.

–getimage10
GetImage 10x10 square.

–getimage100
GetImage 100x100 square.

–getimage500
GetImage 500x500 square.

–noop X protocol NoOperation.

–atom GetAtomName.

–prop GetProperty.

–gc Change graphics context.

–create Create child window and map using MapSubwindows.

–ucreate
Create unmapped window.

–map Map child window via MapWindow on parent.

–unmap
Unmap child window via UnmapWindow on parent.

–destroy
Destroy child window via DestroyWindow parent.

–popup Hide/expose window via Map/Unmap popup window.

–move Move window.

–umove Moved unmapped window.

–movetree
Move window via MoveWindow on parent.

–resize Resize window.

–uresize
Resize unmapped window.

–circulate
Circulate lowest window to top.

–ucirculate
Circulate unmapped window to top.

X DEFAULTS
There are no X defaults used by this program.

SEE ALSO
X, *x11perfcomp*

COPYRIGHT
Copyright 1988, 1989 Digital Equipment Corporation.

See *X* for a full statement of rights and permissions.

AUTHORS

Joel McCormack
Phil Karlton
Susan Angebranndt
Chris Kent

NAME

x11perfcomp – X11 server performance comparison program

SYNOPSIS

`x11perfcomp [-r|-ro] [-l label_file] files`

DESCRIPTION

The *x11perfcomp* program merges the output of several *x11perf* runs into a nice tabular format. It takes the results in each file, fills in any missing test results if necessary, and for each test shows the objects / second rate of each server. If invoked with the **–r** or **–ro** options, it shows the relative performance of each server to the first server.

Normally, *x11perfcomp* uses the first file specified to determine which specific tests it should report on. Some (non-DEC :) servers may fail to perform all tests. In this case, *x11perfcomp* automatically substitutes in a rate of 0.0 objects / second. Since the first file determines which tests to report on, this file must contain a superset of the tests reported in the other files, else *x11perfcomp* will fail.

You can provide an explicit list of tests to report on by using the -l switch to specify a file of labels. You can create a label file by using the -label option in *x11perf*.

OPTIONS

x11perfcomp accepts the options listed below:

–r Specifies that the output should also include relative server performance.

–ro Specifies that the output should include only relative server performance.

–l *label_file*
Specifies a label file to use.

X DEFAULTS

There are no X defaults used by this program.

SEE ALSO

X, *x11perf*

AUTHORS

Mark Moraes wrote the original scripts to compare servers.
Joel McCormack just munged them together a bit.

NAME

xbiff - mailbox flag for X

SYNOPSIS

`xbiff [-toolkitoption ...] [-option ...]`

DESCRIPTION

The *xbiff* program displays a little image of a mailbox. When there is no mail, the flag on the mailbox is down. When mail arrives, the flag goes up and the mailbox beeps. By default, pressing any mouse button in the image forces *xbiff* to remember the current size of the mail file as being the ''empty'' size and to lower the flag.

This program is nothing more than a wrapper around the Athena *Mailbox* widget.

OPTIONS

Xbiff accepts all of the standard X Toolkit command line options along with the additional options listed below:

–file *filename*
This option specifies the name of the file which should be monitored. By default, it watches **$OPENNT_ROOT/usr/spool/mail/***username*, where *username* is your login name.

–help This option indicates that a brief summary of the allowed options should be printed on the standard error.

–shape
This option indicates that the mailbox window should be shaped if masks for the empty or full images are given.

–update *seconds*
This option specifies the frequency in seconds at which *xbiff* should update its display. If the mailbox is obscured and then exposed, it will be updated immediately. The default is 30 seconds.

–volume *percentage*
This option specifies how loud the bell should be rung when new mail comes in.

The following standard X Toolkit command line arguments are commonly used with *xbiff:*

–display *display*
This option specifies the X server to contact.

–geometry *geometry*
This option specifies the preferred size and position of the mailbox window. The mailbox is 48 pixels wide and 48 pixels high and will be centered in the window.

–bg *color*
This option specifies the color to use for the background of the window.

–bd *color*
This option specifies the color to use for the border of the window.

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–fg *color*
This option specifies the color to use for the foreground of the window.

–rv
This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

The application class name is XBiff. This program uses the *Mailbox* widget in the Athena widget set. It understands all of the core resource names and classes as well as:

checkCommand (class **CheckCommand)**
Specifies a shell command to be executed to check for new mail rather than examining the size of *file*. he specified string value is used as the argument to a *system*( ) call and may therefore contain I/O redirection. An exit status of 0 indicates that new mail is waiting, 1 indicates that there has been no change in size, and 2 indicates that the mail has been cleared. By default, no shell command is provided.

file (class **File)**
Specifies the name of the file to monitor. The default is to watch **$OPENNT_ROOT/usr/spool/mail/***username, where username is your login name.*

onceOnly (*class* **Boolean)**
Specifies that the bell is only rung the first time new mail is found and is not rung again until at least one interval has passed with no mail waiting. The window will continue to indicate the presence of new mail until it has been retrieved. The default is false.

width (class **Width)**
Specifies the width of the mailbox.

height (class **Height)**
Specifies the height of the mailbox.

update (class **Interval)**
Specifies the frequency in seconds at which the mail should be checked. The default is 30.

volume (class **Volume)**
Specifies how loud the bell should be rung. The default is 33 percent.

foreground (class **Foreground)**
Specifies the color for the foreground.

reverseVideo (class **ReverseVideo)**
Specifies that the foreground and background should be reversed.

flip (class **Flip)**
Specifies whether or not the image that is shown when mail has arrived should be inverted. The default is ''true.''

fullPixmap (class **Pixmap)**
Specifies a bitmap to be shown when new mail has arrived. The default is flagup.

emptyPixmap (class **Pixmap)**
Specifies a bitmap to be shown when no new mail is present. The default is flagdown.

shapeWindow (class **ShapeWindow)**
Specifies whether or not the mailbox window should be shaped to the given fullPixmapMask and emptyPixmapMask. The default is false.

fullPixmapMask (class **PixmapMask)**
Specifies a mask for the bitmap to be shown when new mail has arrived. The default is none.

emptyPixmapMask (class **PixmapMask)**
Specifies a mask for the bitmap to be shown when no new mail is present. The default is none.

ACTIONS

The *Mailbox* widget provides the following actions for use in event translations:

check() This action causes the widget to check for new mail and display the flag appropriately.
unset() This action causes the widget to lower the flag until new mail comes in.
set() This action causes the widget to raise the flag until the user resets it.

The default translation is

```
<ButtonPress>:  unset()
```

ENVIRONMENT VARIABLES

DISPLAY
to get the default host and display number.
XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X, *xrdb*

stat( )

BUGS

The mailbox bitmaps are ugly.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium
Additional hacks by Ralph Swick, DEC / MIT Project Athena

NAME

xcalc – scientific calculator for X

SYNOPSIS

`xcalc [-stipple] [-rpn] [-toolkitoption...]`

DESCRIPTION

Xcalc is a scientific calculator desktop accessory that can emulate a TI-30 or an HP-10C.

OPTIONS

Xcalc accepts all of the standard toolkit command line options along with two additional options:

–rpn This option indicates that Reverse Polish Notation should be used. In this mode the calculator will look and behave like an HP-10C. Without this flag, it will emulate a TI-30.

–stipple
This option indicates that the background of the calculator should be drawn using a stipple of the foreground and background colors. On monochrome displays improves the appearance.

OPERATION

Pointer Usage: Operations may be performed with pointer button 1, or in some cases, with the keyboard. Many common calculator operations have keyboard accelerators. To quit, press pointer button 3 on the AC key of the TI calculator, or the ON key of the HP calculator.

The numbered keys, the +/- key, and the +, -, *, /, and = keys all do exactly what you would expect them to. It should be noted that the operators obey the standard rules of precedence. Thus, entering "3+4*5=" results in "23", not "35". The parentheses can be used to override this. For example, "(1+2+3)*(4+5+6)=" results in "6*15=90".

The entire number in the calculator display can be selected, in order to paste the result of a calculation into text.

The action procedures associated with each function are given below. These are useful if you are interested in defining a custom calculator. The action used for all digit keys is **digit(*n*)**, where *n* is the corresponding digit, 0..9.

1/x Replaces the number in the display with its reciprocal. The corresponding action procedure is **reciprocal()**.

x^2 Squares the number in the display. The corresponding action procedure is **square()**.

SQRT Takes the square root of the number in the display. The corresponding action procedure is **squareRoot()**.

CE/C When pressed once, clears the number in the display without clearing the state of the machine. Allows you to re-enter a number if you make a mistake. Pressing it twice clears the state, also. The corresponding action procedure for TI mode is **clear()**.

AC Clears the display, the state, and the memory. Pressing it with the third pointer button turns off the calculator, in that it exits the program. The action procedure to clear the state is **off()**; to quit,

quit().

INV Invert function. See the individual function keys for details. The corresponding action procedure is **inverse()**.

sin Computes the sine of the number in the display, as interpreted by the current DRG mode (see DRG, below). If inverted, it computes the arcsine. The corresponding action procedure is **sine()**.

cos Computes the cosine, or arccosine when inverted. The corresponding action procedure is **cosine()**.

tan Computes the tangent, or arctangent when inverted. The corresponding action procedure is **tangent()**.

DRG Changes the DRG mode, as indicated by 'DEG', 'RAD', or 'GRAD' at the bottom of of the calculator ''liquid crystal'' display. When in 'DEG' mode, numbers in the display are taken as being degrees. In 'RAD' mode, numbers are in radians, and in 'GRAD' mode, numbers are in grads. When inverted, the DRG key has a feature of converting degrees to radians to grads and vice-versa. Example: put the calculator into 'DEG' mode, and enter "45 INV DRG". The display should now show something along the lines of ".785398", which is 45 degrees converted to radians. The corresponding action procedure is **degree()**.

e The constant 'e'. (2.7182818...). The corresponding action procedure is **e()**.

EE Used for entering exponential numbers. For example, to get "-2.3E-4" you'd enter "2 . 3 + / - EE 4 + / -". The corresponding action procedure is **scientific()**.

log Calculates the log (base 10) of the number in the display. When inverted, it raises "10.0" to the number in the display. For example, entering "3 INV log" should result in "1000". The corresponding action procedure is **logarithm()**.

ln Calculates the log (base e) of the number in the display. When inverted, it raises "e" to the number in the display. For example, entering "e ln" should result in "1". The corresponding action procedure is **naturalLog()**.

y^x Raises the number on the left to the power of the number on the right. For example "2 y^x 3 =" results in "8", which is 2^3. For a further example, "(1+2+3) y^x (1+2) =" equals "6 y^x 3" which equals "216". The corresponding action procedure is **power()**.

PI The constant 'pi'. (3.1415927....) The corresponding action procedure is **pi()**.

x! Computes the factorial of the number in the display. The number in the display must be an integer in the range 0-500, though, depending on your math library, it might overflow long before that. The corresponding action procedure is **factorial()**.

(Left parenthesis. The corresponding action procedure for TI calculators is **leftParen()**.

) Right parenthesis. The corresponding action procedure for TI calculators is **rightParen()**.

/ Division. The corresponding action procedure is **divide()**.

***** Multiplication. The corresponding action procedure is **multiply()**.

– Subtraction. The corresponding action procedure is **subtract()**.

+ Addition. The corresponding action procedure is **add()**.

= Perform calculation. The TI-specific action procedure is **equal()**.

STO Copies the number in the display to the memory location. The corresponding action procedure is **store()**.

RCL Copies the number from the memory location to the display. The corresponding action procedure is **recall()**.

SUM Adds the number in the display to the number in the memory location. The corresponding action procedure is **sum()**.

EXC Swaps the number in the display with the number in the memory location. The corresponding action procedure for the TI calculator is **exchange()**.

+/- Negate; change sign. The corresponding action procedure is **negate()**.

. Decimal point. The action procedure is **decimal()**.

Calculator Key *Usage* (RPN *mode):* The number keys, CHS (change sign), +, -, *, /, and ENTR keys all do exactly what you would expect them to do. Many of the remaining keys are the same as in TI mode. The differences are detailed below. The action procedure for the ENTR key is **enter()**.

<- This is a backspace key that can be used if you make a mistake while entering a number. It will erase digits from the display. (See BUGS). Inverse backspace will clear the X register. The corresponding action procedure is **back()**.

ON Clears the display, the state, and the memory. Pressing it with the third pointer button turns off the calculator, in that it exits the program. To clear state, the action procedure is **off**; to quit, **quit()**.

INV Inverts the meaning of the function keys. This would be the *f* key on an HP calculator, but *xcalc* does not display multiple legends on each key. See the individual function keys for details.

10^x Raises "10.0" to the number in the top of the stack. When inverted, it calculates the log (base 10) of the number in the display. The corresponding action procedure is **tenpower()**.

e^x Raises "e" to the number in the top of the stack. When inverted, it calculates the log (base e) of the number in the display. The action procedure is **epower()**.

STO Copies the number in the top of the stack to a memory location. There are 10 memory locations. The desired memory is specified by following this key with a digit key.

RCL Pushes the number from the specified memory location onto the stack.

SUM Adds the number on top of the stack to the number in the specified memory location.

x:y Exchanges the numbers in the top two stack positions, the X and Y registers. The corresponding action procedure is **XexchangeY()**.

R v Rolls the stack downward. When inverted, it rolls the stack upward. The corresponding action procedure is **roll()**.

blank These keys were used for programming functions on the HP-10C. Their functionality has not been duplicated in *xcalc*.

Finally, there are two additional action procedures: **bell()**, which rings the bell; and **selection()**, which performs a cut on the entire number in the calculator's ''liquid crystal'' display.

ACCELERATORS

Accelerators are shortcuts for entering commands. *xcalc* provides some sample keyboard accelerators; also users can customize accelerators. The numeric keypad accelerators provided by *xcalc* should be intuitively correct. The accelerators defined by *xcalc* on the main keyboard are given below:

TI Key	HP Key	Keyboard Accelerator	TI Function	HP Function
SQRT	SQRT	r	squareRoot()	squareRoot()
AC	ON	space	clear()	clear()
AC	<-	Delete	clear()	back()
AC	<-	Backspace	clear()	back()
AC	<-	Control-H	clear()	back()
AC	Clear	clear()		
AC	ON	q	quit()	quit()
AC	ON	Control-C	quit()	quit()
INV	i	i	inverse()	inverse()
sin	s	s	sine()	sine()
cos	c	c	cosine()	cosine()
tan	t	t	tangent()	tangent()
DRG	DRG	d	degree()	degree()
e	e	e()		
ln	ln	l	naturalLog()	naturalLog()
y^x	y^x	^	power()	power()
PI	PI	p	pi()	pi()
x!	x!	!	factorial()	factorial()
(	(		leftParen()	
)	)		rightParen()	
/	/	/	divide()	divide()
*	*	*	multiply()	multiply()
-	-	-	subtract()	subtract()
+	+	+	add()	add()
=	=		equal()	
0..9	0..9	0..9	digit()	digit()
+/-	CHS	n	negate()	negate()
	x:y	x	XexchangeY()	
	ENTR	Return	enter()	
	ENTR	Linefeed	enter()	

CUSTOMIZATION

The application class name is XCalc.

Xcalc has an enormous application defaults file which specifies the position, label, and function of each key on the calculator. It also gives translations to serve as keyboard accelerators. Because these resources are not specified in the source code, you can create a customized calculator by writing a private

application defaults file, using the Athena Command and Form widget resources to specify the size and position of buttons, the label for each button, and the function of each button.

The foreground and background colors of each calculator key can be individually specified. For the TI calculator, a classical color resource specification might be:

```
XCalc.ti.Command.background:      gray50
XCalc.ti.Command.foreground:      white
```

For each of buttons 20, 25, 30, 35, and 40, specify:

```
XCalc.ti.button20.background:     black
XCalc.ti.button20.foreground:     white
```

For each of buttons 22, 23, 24, 27, 28, 29, 32, 33, 34, 37, 38, and 39:

```
XCalc.ti.button22.background:     white
XCalc.ti.button22.foreground:     black
```

WIDGET HIERARCHY

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xcalc*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XCalc xcalc
   Form  ti  or  hp     (the name depends on the mode)
      Form  bevel
         Form  screen
            Label  M
            Toggle  LCD
            Label  INV
            Label  DEG
            Label  RAD
            Label  GRAD
            Label  P
      Command  button1
      Command  button2
      Command  button3
and so on, ...
      Command  button38
      Command  button39
      Command  button40
```

APPLICATION RESOURCES

rpn (Class **Rpn**)
: Specifies that the rpn mode should be used. The default is TI mode.

stipple (Class **Stipple**)
: Indicates that the background should be stippled. The default is ‘‘on’’ for monochrome displays, and ‘‘off’’ for color displays.

cursor (Class **Cursor**)
: The name of the symbol used to represent the pointer. The default is

''hand2''.

COLORS

If you would like xcalc to use its ti colors, include the following in the #ifdef COLOR section of the file you read with *xrdb*:

```
*customization:                    -color
```

This will cause xcalc to pick up the colors in the app-defaults color customization file: **$OPENNT_ROOT/usr/lib/X11/app-defaults/XCalc-color**.

SEE ALSO

X, *xrdb*, the Athena Widget Set

BUGS

HP mode: A bug report claims that the sequence of keys 5, ENTER, <- should clear the display, but it doesn't.

COPYRIGHT

Copyright 1988, 1989, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHORS

John Bradley, University of Pennsylvania
Mark Rosenstein, MIT Project Athena
Donna Converse, MIT X Consortium

NAME

xclipboard - X clipboard client

SYNOPSIS

`xclipboard [-`*`toolkitoption`* `...] [-w] [-nw]`

DESCRIPTION

The *xclipboard* program is used to collect and display text selections that are sent to the CLIPBOARD by other clients. It is typically used to save CLIPBOARD selections for later use. It stores each CLIPBOARD selection as a separate string, each of which can be selected. Each time CLIPBOARD is asserted by another application, *xclipboard* transfers the contents of that selection to a new buffer and displays it in the text window. Buffers are never automatically deleted, so you'll want to use the delete button to get rid of useless items.

Since *xclipboard* uses a Text Widget to display the contents of the clipboard, text sent to the CLIPBOARD may be re-selected for use in other applications. *xclipboard* also responds to requests for the CLIPBOARD selection from other clients by sending the entire contents of the currently displayed buffer.

An *xclipboard* window has the following buttons across the top:

quit When this button is pressed, *xclipboard* exits.
delete When this button is pressed, the current buffer is deleted and the next one displayed.
new Creates a new buffer with no contents. Useful in constructing a new CLIPBOARD selection by hand.
next Displays the next buffer in the list.
previous Displays the previous buffer.

OPTIONS

The *xclipboard* program accepts all of the standard X Toolkit command line options as well as the following:

–nw This option indicates that long lines of text should not wrap around. This is the default behavior.

–w This option indicates that lines of text that are too long to be displayed on one line in the clipboard should wrap around to the following lines.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xclipboard*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XClipboard  xclipboard
   Form  form
      Command  quit
      Command  delete
      Command  new
      Command  next
```

```
Command  prev
Text  text
```

SENDING/RETRIEVING CLIPBOARD CONTENTS

Text is copied to the clipboard whenever a client asserts ownership of the **CLIPBOARD** selection. Text is copied from the clipboard whenever a client requests the contents of the **CLIPBOARD** selection. Examples of event bindings that a user may wish to include in a resource configuration file to use the clipboard are:

```
*VT100.Translations: #override \
   <Btn3Up>:      select-end(CLIPBOARD) \n\
   <Btn2Up>:      insert-selection(PRIMARY,CLIPBOARD) \n\
   <Btn2Down>:    ignore ()
```

SEE ALSO

X, *xcutsel*, *xterm*, individual client documentation for how to make a selection and send it to the CLIPBOARD.

ENVIRONMENT VARIABLES

DISPLAY
: Contains the default host and display number.

XENVIRONMENT
: Contains the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

FILES

$OPENNT_ROOT/usr/lib/X11/app-defaults/XClipboard
: Specifies required resources.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology
See *X* for a full statement of rights and permissions.

AUTHOR

Ralph R. Swick, DEC / MIT Project Athena
Chris D. Peterson, MIT X Consortium
Keith Packard, MIT X Consortium

NAME

xclock - analog / digital clock for X

SYNOPSIS

```
xclock [–toolkitoption ...] [-analog] [-chime] [-digital]
       [-help] [-hd color] [hl color] [padding number]
       [update seconds]
```

DESCRIPTION

The *xclock* program displays the time in analog or digital form. The time is continuously updated at a frequency which may be specified by the user. This program is nothing more than a wrapper around the Athena Clock widget.

OPTIONS

Xclock accepts all of the standard X Toolkit command line options along with the additional options listed below:

–help This option indicates that a brief summary of the allowed options should be printed on the standard error.

–analog
This option indicates that a conventional 12 hour clock face with tick marks and hands should be used. This is the default. If both **–analog** and **–digital** are specified, the last one on the command line takes effect.

–chime
This option indicates that the clock should chime once on the half hour and twice on the hour.

–digital
This option indicates that a 24 hour digital clock should be used. If both **–analog** and **–digital** are specified, the last one on the command line takes effect.

–hd *color*
This option specifies the color of the hands on an analog clock. The default is **black**.

–hl *color*
This option specifies the color of the edges of the hands on an analog clock, and is only useful on color displays. The default is **black**.

–padding *number*
This option specifies the width in pixels of the padding between the window border and clock text or picture. The default is 10 on a digital clock and 8 on an analog clock.

–update *seconds*
This option specifies the frequency in seconds at which *xclock* should update its display. If the clock is obscured and then exposed, it will be updated immediately. A value of less than 30 seconds will enable a second hand on an analog clock. The default is 60 seconds.

X DEFAULTS

This program uses the Athena *Clock* widget. It understands all of the core

resource names and classes as well as:

width (class **Width)**
Specifies the width of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

height (class **Height)**
Specifies the height of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

update (class **Interval)**
Specifies the frequency in seconds at which the time should be redisplayed.

foreground (class **Foreground)**
Specifies the color for the tic marks. The default is depends on whether *reverseVideo* is specified. If *reverseVideo* is specified the default is *lwhite*, otherwise the default is **black**.

hands (class **Foreground)**
Specifies the color of the insides of the clock's hands. The default is depends on whether *reverseVideo* is specified. If *reverseVideo* is specified the default is *lwhite*, otherwise the default is **black**.

highlight (class **Foreground)**
Specifies the color used to highlight the clock's hands. The default is depends on whether *reverseVideo* is specified. If *reverseVideo* is specified the default is *lwhite*, otherwise the default is **black**.

analog (class **Boolean)**
Specifies whether or not an analog clock should be used instead of a digital one. The default is True.

chime (class **Boolean)**
Specifies whether or not a bell should be rung on the hour and half hour.

padding (class **Margin)**
Specifies the amount of internal padding in pixels to be used. The default is 8.

font (class **Font)**
Specifies the font to be used for the digital clock. Note that variable width fonts currently will not always display correctly.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xclock*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XClock  xclock
   Clock  clock
```

ENVIRONMENT VARIABLES

DISPLAY
Contains the default host and display number.

XENVIRONMENT
Contains the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

FILES

$OPENNT_ROOT/usr/lib/X11/app-defaults/XClock
Specifies required resources.

SEE ALSO

X, *xrdb*

time( ), Athena Clock widget

BUGS

Xclock believes the system clock.

When in digital mode, the string should be centered automatically.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHORS

Tony Della Fera (MIT-Athena, DEC)
Dave Mankins (MIT-Athena, BBN)
Ed Moy (UC Berkeley)

NAME

xcmsdb – Xlib Screen Color Characterization Data utility

SYNOPSIS

xcmsdb **[**–*option ...*] [*filename*]

DESCRIPTION

xcmsdb is used to load, query, or remove Screen Color Characterization Data stored in properties on the root window of the screen. Screen Color Characterization Data is an integral part of Xlib, necessary for proper conversion between device-independent and device-dependent color specifications. Xlib uses the XDCCC_LINEAR_RGB_MATRICES and XDCCC_LINEAR_RGB_CORRECTION properties to store color characterization data for color monitors. It uses XDCCC_GRAY_SCREENWWHITEPOINT and XDCCC_GRAY_CORRECTION properties for gray scale monitors. Because Xlib allows the addition of Screen Color Characterization Function Sets, added function sets may place their Screen Color Characterization Data on other properties. This utility is unaware of these other properties, therefore, you will need to use a similar utility provided with the function set, or use the *xprop* utility.

The ASCII readable contents of *filename* (or the standard input if no input file is given) are appropriately transformed for storage in properties, provided the **–query** or **–remove** options are not specified.

OPTIONS

xcmsdb program accepts the following options:

–color This option sets the query and remove options to only check for the XDCCC_LINEAR_RGB_MATRICES and XDCCC_LINEAR_RGB_CORRECTION properties. If the **–color** option is not set then the query and remove options check for all the properties.

–format 32 | **16** | **8**
Specifies the property format (32, 16, or 8 bits per entry) for the XDCCC_LINEAR_RGB_CORRECTION property. Precision of encoded floating point values increases with the increase in bits per entry. The default is 32 bits per entry.

–query
This option attempts to read the XDCCC properties off the screen's root window. If successful, it transforms the data into a more readable format, then sends the data to standard out.

–remove
This option attempts to remove the XDCCC properties on the screen's root window.

SEE ALSO

xprop, Xlib documentation

ENVIRONMENT VARIABLES

DISPLAY
Contains which display and screen to use.

BUGS
Unknown.

COPYRIGHT
Copyright 1990, Tektronix Inc.

AUTHOR
Chuck Adams, Tektronix Inc.

NAME

xcmstest – XCMS test program

SYNOPSIS

```
xcmstest [-display displayname]
xcmstest -echo [-display displayname] < cmdfile
```

DESCRIPTION

xcmstest was developed as a simple test interface for testing the XCMS API library during its development.

BEFORE INVOCATION

Insure that the Screen Color Characterization Data has been placed in the appropriate properties on the screen's root window. To load these properties, refer to the xcmsdb utility provided with this release.

In order to convert color names using a client-side color name database, the database file must be specified via the XCMSDB environment variable.

To use the sample database provided with this release set XCMSDB as follows:

```
$ setenv XCMSDB 'pwd'/Xcms.txt
```

USING XCMSTEST

Upon invocation of *xcmstest*, you should receive a "XCMS >" prompt. At this point you can begin entering commands.

To list the commands currently available enter **list**:

```
XCMS >  list
```

To receive help for a specific command enter the command name followed by **–help**. For example:

```
XCMS >  LookupColor -help
```

The **–echo** option provides a more readable output, echoing the command from the cmdfile. At some point in time, you might try using some simple test cases provided with the release (in the **tc** directory, e.g. **test1**, **test2**, **AllocNColor**, gamutquery**).**For file **test1**, invoke *xcmstest* as follows:

```
$ xcmstest -echo < tc/test1
```

CAVEATS

Color name strings passed to *xcmstest* commands cannot contain a space. However, since spaces are ignored by the XCMS API library, you can pass a color name with the spaces eliminated. For example:

```
XCMS >  LookupColor cornflowerblue UNDEFINED RGB
```

SEE ALSO

xcmsdb, Chapter 6 of *Xlib — C Language X Interface*.

COPYRIGHT

Copyright 1991, Massachusetts Institute of Technology.

NAME

xcutsel - interchange between cut buffer and selection

SYNOPSIS

`xcutsel`
`[-toolkitoption ...] [-selection` *`selection`*`] [-cutbuffer` *`number`*`]`

DESCRIPTION

The *xcutsel* program is used to copy the current selection into a cut buffer and to make a selection that contains the current contents of the cut buffer. It acts as a bridge between applications that don't support selections and those that do.

By default, *xcutsel* will use the selection named PRIMARY and the cut buffer CUT_BUFFER0. Either or both of these can be overridden by command line arguments or by resources.

An *xcutsel* window has the following buttons:

quit
: When this button is pressed, *xcutsel* exits. Any selections held by *xcutsel* are automatically released.

copy PRIMARY to 0
: When this button is pressed, *xcutsel* copies the current selection into the cut buffer.

copy 0 to PRIMARY
: When this button is pressed, *xcutsel* converts the current contents of the cut buffer into the selection.

The button labels reflect the selection and cutbuffer selected by command line options or through the resource database.

When the *copy 0 to PRIMARY* button is activated, the button will remain inverted as long as *xcutsel* remains the owner of the selection. This serves to remind you which client owns the current selection. Note that the value of the selection remains constant; if the cutbuffer is changed, you must again activate the copy button to retrieve the new value when desired.

OPTIONS

Xcutsel accepts all of the standard X Toolkit command line options as well as the following:

–cutbuffer *number*
: Use cut buffer *number*. The default is cut buffer 0.

–selection *name*
: Use selection *name*. The default is PRIMARY. The only supported abbreviations for this option are **–select**, **–sel** and **–s**, because the standard toolkit option **–selectionTimeout** has a similar name.

X DEFAULTS

This program accepts all of the standard X Toolkit resource names and classes as well as:

selection (class **Selection)**
: This resource specifies the name of the selection to use. The default is PRIMARY.

cutBuffer (class **CutBuffer)**
This resource specifies the number of the cut buffer to use. The default is 0.

WIDGET NAMES
The following instance names may be used when user configuration of the labels in them is desired:

sel-cut (class **Command)**
This is the *copy SELECTION to BUFFER* button.

cut-sel (class **Command)**
This is the *copy BUFFER to SELECTION* button.

quit (class **Command)**
This is the *quit* button.

SEE ALSO
X, *xclipboard*, *xterm*, text widget documentation, individual client documentation for how to make a selection.

BUGS
There is no way to change the name of the selection or the number of the cut buffer while the program is running.

COPYRIGHT
Copyright 1988, Massachusetts Institute of Technology

See *X* for a full statement of rights and permissions.

AUTHOR
Ralph R. Swick, DEC / MIT Project Athena

NAME

xditview – display ditroff output

SYNOPSIS

`xditview [-toolkitoption ...] [-option ...] [filename]`

DESCRIPTION

The *xditview* program displays *ditroff* output on an X display. It uses no special metrics and automatically converts the printer coordinates into screen coordinates; using the user-specified screen resolution, rather than the actual resolution so that the appropriate fonts can be found. If **–** is given as the *filename, xditview* reads from standard input. If ''**|**'' is the first character of *filename, xditview* forks *sh* to run the rest of the ''file name'' and uses the standard output of that command.

OPTIONS

Xditview accepts all of the standard X Toolkit command line options along with the additional options listed below:

–page *page-number*
This option specifies the page number of the document to be displayed at start up time.

–resolution *screen-resolution*
This specifies the desired screen resolution to use; fonts will be opened by requesting this resolution field in the XLFD names.

–noPolyText
Some X servers incorrectly implement PolyText with multiple strings per request. This option suppesses the use of this feature in *xditview*.

–backingStore *backing-store-type*
Redisplay can take up to a second or so; this option causes the server to save the window contents so that when it is scrolled around the viewport, the window is painted from contents saved in backing store. *backing-store-type* can be one of **Always**, **WhenMapped** or **NotUseful**.

The following standard X Toolkit command line arguments are commonly used with *xditview:*

–bg *color*
This option specifies the color to use for the background of the window. The default is **white**.

–bd *color*
This option specifies the color to use for the border of the window. The default is **black**.

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–display *host:display*
This option specifies the X server to contact.

–fg *color*
This option specifies the color to use for displaying text. The default is **black**.

–fn *font*
This option specifies the font to be used for displaying widget text. The default is **fixed**.

–geometry *geometry*
This option specifies the preferred size and position of the window.

–rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–xrm *resourcestring*
This option specifies a resource string to be used.

X DEFAULTS

This program uses a *Dvi* widget. It understands all of the core resource names and classes as well as:

width (class **Width)**
Specifies the width of the window.

height (class **Height)**
Specifies the height of the window.

foreground (class **Foreground)**
Specifies the default foreground color.

font (class **Font)**
Specifies the font to be used for error messages.

FontMap (class FontMap)
To associate the *ditroff* fonts with appropriate X fonts, this string resource contains a set of new-line separated specifications, each of which consists of a ditroff name, some white space and an XLFD pattern with * characters in appropriate places to allow all sizes to be listed. The default fontMap is:

```
R  –*–times–medium–r–normal– –*–*–*–*–*–*–iso8859–1\n\
I  –*–times–medium–i–normal– –*–*–*–*–*–*–iso8859–1\n\
B  –*–times–bold–r–normal– –*–*–*–*–*–*–iso8859–1\n\
F  –*–times–bold–i–normal– –*–*–*–*–*–*–iso8859–1\n\
BI  –*–times–bold–i–normal– –*–*–*–*–*–*–iso8859–1\n\
C  –*–courier–medium–r–normal– –*–*–*–*–*–*–iso8859–1\n\
CO  –*–courier–medium–o–normal– –*–*–*–*–*–*–iso8859–1\n\
CB  –*–courier–bold–r–normal– –*–*–*–*–*–*–iso8859–1\n\
CF  –*–courier–bold–o–normal– –*–*–*–*–*–*–iso8859–1\n\
H  –*–helvetica–medium–r–normal– –*–*–*–*–*–*–iso8859–1\n\
HO  –*–helvetica–medium–o–normal– –*–*–*–*–*–*–iso8859–1\n\
HB  –*–helvetica–bold–r–normal– –*–*–*–*–*–*–iso8859–1\n\
HF  –*–helvetica–bold–o–normal– –*–*–*–*–*–*–iso8859–1\n\
N  –*–new century schoolbook–medium–r–normal– –*–*–*–*–*–*–iso8859–1\n\
```

```
NI  –*–new century schoolbook–medium–i–normal– –*–*–*–*–*–*–iso8859–1\n\
NB  –*–new century schoolbook–bold–r–normal– –*–*–*–*–*–*–iso8859–1\n\
NF  –*–new century schoolbook–bold–i–normal– –*–*–*–*–*–*–iso8859–1\n\
A  –*–charter–medium–r–normal– –*–*–*–*–*–*–iso8859–1\n\
AI  –*–charter–medium–i–normal– –*–*–*–*–*–*–iso8859–1\n\
AB  –*–charter–bold–r–normal– –*–*–*–*–*–*–iso8859–1\n\
AF  –*–charter–bold–i–normal– –*–*–*–*–*–*–iso8859–1\n\
S  –*–symbol–medium–r–normal– –*–*–*–*–*–*–adobe–fontspecific\n\
S2  –*–symbol–medium–r–normal– –*–*–*–*–*–*–adobe–fontspecific\n
```

USING XDITVIEW WITH DITROFF

You can use any ditroff output file with *xditview,* although files which use the fonts appropriate to the fontMap will look more accurate on the screen. On servers which support scaled fonts, all requested font sizes will be accurately reflected on the screen; for servers which do not support scaled *xditview* will use the closest font from the same family.

SEE ALSO

X, xrdb, ditroff, X Logical Font Description Conventions

ORIGIN

Portions of this program originated in *xtroff* which was derived from *suntroff.*

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHORS

Keith Packard (MIT X Consortium)
Richard L. Hyde (Purdue)
David Slattengren (Berkeley)
Malcom Slaney (Schlumberger Palo Alto Research)
Mark Moraes (University of Toronto)

NAME

xdpr – dump an X window directly to a printer

SYNOPSIS

```
xdpr [filename] [-display host:display] [-Pprinter]
     [-device devtype] [option . . .]
```

DESCRIPTION

Xdpr uses the commands *xwd*, *xpr*, and *lpr* or *lp* to dump an X window, process it for a particular printer type, and print it out on the printer of your choice. This is the easiest way to get a printout of a window. *Xdpr* by default will print the largest possible representation of the window on the output page.

The options for *xdpr* are the same as those for *xpr*, *xwd*, and *lpr* or *lp*. The most commonly-used options are described below; see the manual pages for these commands for more detailed descriptions of the many options available. The optional *filename* contains a window dump created by *xwd* to be printed instead of selecting an X window.

Xdpr takes these options:

–device *devtype*

Specifies the device on which the file will be printed. Currently supported:

la100	Digital LA100
ljet	HP LaserJet series and other monochrome PCL devices such as ThinkJet, QuietJet, RuggedWriter, HP2560 series, and HP2930 series printers
ln03	Digital LN03
pjet	HP PaintJet (color mode)
pjetxl	HP PaintJet XL Color Graphics Printer (color mode)
pp	IBM PP3812
ps	PostScript printer

The default is PostScript. The option **–device lw** (LaserWriter) is equivalent to **–device ps** and is provided only for backwards compatibility.

–display *host:display*[**.***screen*]

Normally, *xdpr* gets the host and display number to use from the environment variable *DISPLAY*. One can, however, specify them explicitly; see *X*.

–help

This option displays the list of options known to *xdpr*.

–P*printer*

Specifies a printer to send the output to. If a printer name is not specified here, *xdpr* (really *lp*) will send your output to the printer specified by the *PRINTER* environment variable. Be sure that type of the printer matches the type specified with the **–device** option.

Any other arguments will be passed to the *xwd*, *xpr*, and *lp*.

ENVIRONMENT VARIABLES

Xdpr makes use of the following environment variables, if set:

DISPLAY
: which display to use by default.

PRINTER
: which printer to use by default.

COPYRIGHT

Copyright 1985, 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHORS

Paul Boutin, MIT Project Athena
Michael R. Gretzinger, MIT Project Athena
Jim Gettys, MIT Project Athena commands as appropriate for each.

SEE ALSO

lp, *xwd*, *xpr*, *xwud*, *X*

NAME

xdpyinfo – display information utility for X

SYNOPSIS

xdpyinfo **[**–display *displayname***]**

DESCRIPTION

Xdpyinfo is a utility for displaying information about an X server. It is used to examine the capabilities of a server, the predefined values for various parameters used in communicating between clients and the server, and the different types of screens and visuals that are available.

EXAMPLE

The following shows a sample produced by *xdpyinfo* when connected to display that supports an 8 plane screen and a 1 plane screen.

```
name of display:    :0.0
version number:    11.0
vendor string:    MIT X Consortium
vendor release number:    4
maximum request size:  16384 longwords (65536 bytes)
motion buffer size:  0
bitmap unit, bit order, padding:    32, MSBFirst, 32
image byte order:    MSBFirst
number of supported pixmap formats:    2
supported pixmap formats:
    depth 1, bits_per_pixel 1, scanline_pad 32
    depth 8, bits_per_pixel 8, scanline_pad 32
keycode range:    minimum 8, maximum 129
focus:  PointerRoot
number of extensions:    4
    SHAPE
    MIT-SHM
    Multi-Buffering
    MIT-SUNDRY-NONSTANDARD
default screen number:    0
number of screens:    2

screen #0:
  dimensions:    1152x900 pixels (325x254 millimeters)
  resolution:    90x90 dots per inch
  depths (2):    1, 8
  root window id:    0x8006e
  depth of root window:    8 planes
  number of colormaps:    minimum 1, maximum 1
  default colormap:    0x8006b
  default number of colormap cells:    256
  preallocated pixels:    black 1, white 0
  options:    backing-store YES, save-unders YES
  current input event mask:    0xd0801d
    KeyPressMask       ButtonPressMask     ButtonReleaseMask
    EnterWindowMask    ExposureMask        SubstructureRedirectMask
```

```
  PropertyChangeMask ColormapChangeMask
number of visuals:    6
default visual id:  0x80065
visual:
  visual id:    0x80065
  class:    PseudoColor
  depth:    8 planes
  size of colormap:    256 entries
  red, green, blue masks:    0x0, 0x0, 0x0
  significant bits in color specification:    8 bits
visual:
  visual id:    0x80066
  class:    DirectColor
  depth:    8 planes
  size of colormap:    8 entries
  red, green, blue masks:    0x7, 0x38, 0xc0
  significant bits in color specification:    8 bits
visual:
  visual id:    0x80067
  class:    GrayScale
  depth:    8 planes
  size of colormap:    256 entries
  red, green, blue masks:    0x0, 0x0, 0x0
  significant bits in color specification:    8 bits
visual:
  visual id:    0x80068
  class:    StaticGray
  depth:    8 planes
  size of colormap:    256 entries
  red, green, blue masks:    0x0, 0x0, 0x0
  significant bits in color specification:    8 bits
visual:
  visual id:    0x80069
  class:    StaticColor
  depth:    8 planes
  size of colormap:    256 entries
  red, green, blue masks:    0x7, 0x38, 0xc0
  significant bits in color specification:    8 bits
visual:
  visual id:    0x8006a
  class:    TrueColor
  depth:    8 planes
  size of colormap:    8 entries
  red, green, blue masks:    0x7, 0x38, 0xc0
  significant bits in color specification:    8 bits
number of mono multibuffer types:    6
  visual id, max buffers, depth:    0x80065, 0, 8
  visual id, max buffers, depth:    0x80066, 0, 8
  visual id, max buffers, depth:    0x80067, 0, 8
  visual id, max buffers, depth:    0x80068, 0, 8
  visual id, max buffers, depth:    0x80069, 0, 8
```

```
    visual id, max buffers, depth:    0x8006a, 0, 8
  number of stereo multibuffer types:    0

screen #1:
  dimensions:    1152x900 pixels (325x254 millimeters)
  resolution:    90x90 dots per inch
  depths (1):    1
  root window id:    0x80070
  depth of root window:    1 plane
  number of colormaps:    minimum 1, maximum 1
  default colormap:    0x8006c
  default number of colormap cells:    2
  preallocated pixels:    black 1, white 0
  options:    backing-store YES, save-unders YES
  current input event mask:    0xd0801d
    KeyPressMask       ButtonPressMask    ButtonReleaseMask
    EnterWindowMask    ExposureMask       SubstructureRedirectMask
    PropertyChangeMask ColormapChangeMask
  number of visuals:    1
  default visual id:  0x80064
  visual:
    visual id:    0x80064
    class:    StaticGray
    depth:    1 plane
    size of colormap:    2 entries
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    1 bits
  number of mono multibuffer types:    1
    visual id, max buffers, depth:    0x80064, 0, 1
  number of stereo multibuffer types:    0
```

ENVIRONMENT VARIABLES

DISPLAY
Contains the default host, display number, and screen.

SEE ALSO

X, *xwininfo*, *xprop*, *xrdb*

COPYRIGHT

Copyright 1988, 1989, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xedit – simple text editor for X

SYNOPSIS

`xedit [-toolkitoption ... ] [filename]`

DESCRIPTION

Xedit provides a window consisting of the following four areas:

Commands Section
: A set of commands that allow you to exit *xedit*, save the file, or load a new file into the edit window.

Message Window
: Displays *xedit* messages. In addition, this window can be used as a scratch pad.

Filename Display Displays the name of the file currently being edited, and whether this file is *Read-Write* or *Read Only*.

Edit Window
: Displays the text of the file that you are editing or creating.

OPTIONS

Xedit accepts all of the standard X Toolkit command line options (see *X*). The order of the command line options is not important.

filename
: Specifies the file that is to be loaded during start-up. This is the file which will be edited. If a file is not specified, *xedit* lets you load a file or create a new file after it has started up.

EDITING

The Athena Text widget is used for the three sections of this application that allow text input. The characters typed will go to the Text widget that the pointer cursor is currently over. If the pointer cursor is not over a text widget then the keypresses will have no effect on the application. This is also true for the special key sequences that popup dialog widgets, so typing Control-S in the filename widget will enable searching in that widget, not the edit widget.

Both the message window and the edit window will create a scrollbar if the text to display is too large to fit in that window. Horizontal scrolling is not allowed by default, but can be turned on through the Text widget's resources, see *Athena Widget Set* for the exact resource definition.

The following keystroke combinations are defined:

Ctrl-a	Beginning Of Line	**Meta-b**	Backward Word
Ctrl-b	Backward Character	**Meta-f**	Forward Word
Ctrl-d	Delete Next Character	**Meta-i**	Insert File
Ctrl-e	End Of Line	**Meta-k**	Kill To End Of Paragraph

Ctrl-f	Forward Character	**Meta-q**	Form Paragraph
Ctrl-g	Multiply Reset	**Meta-v**	Previous Page
Ctrl-h	Delete Previous Character	**Meta-y**	Insert Current Selection
Ctrl-j	Newline And Indent	**Meta-z**	Scroll One Line Down
Ctrl-k	Kill To End Of Line	**Meta-d**	Delete Next Word
Ctrl-l	Redraw Display	**Meta-D**	Kill Word
Ctrl-m	Newline	**Meta-h**	Delete Previous Word
Ctrl-n	Next Line	**Meta-H**	Backward Kill Word
Ctrl-o	Newline And Backup	**Meta-<**	Beginning Of File
Ctrl-p	Previous Line	**Meta->**	End Of File
Ctrl-r	Search / Replace Backward	**Meta-]**	Forward Paragraph
Ctrl-s	Search / Replace Forward	**Meta-[**	Backward Paragraph
Ctrl-t	Transpose Characters		
Ctrl-u	Multiply by 4	**Meta-Delete**	Delete Previous Word
Ctrl-v	Next Page	**Meta-Shift Delete**	Kill Previous Word
Ctrl-w	Kill Selection	**Meta-Backspace**	Delete Previous Word
Ctrl-y	Unkill	**Meta-Shift Backspace**	Kill Previous Word
Ctrl-z	Scroll One Line Up		

In addition, the pointer may be used to cut and paste text:

Button 1 Down	Start Selection
Button 1 Motion	Adjust Selection
Button 1 Up	End Selection (cut)
Button 2 Down	Insert Current Selection (paste)
Button 3 Down	Extend Current Selection
Button 3 Motion	Adjust Selection
Button 3 Up	End Selection (cut)

COMMANDS

Quit — Quits the current editing session. If any changes have not been saved, *xedit* displays a warning message, allowing the user to save the file.

Save — If file backups are enabled (see RESOURCES, below) *xedit* stores a copy of the original, unedited file in <prefix>*file*<suffix>, then overwrites the *file* with the contents of the edit window. The filename is retrieved from the Text widget directly to the right of the *Load* button.

Load — Loads the file named in the text widget immediately to the right of the this button and displays it in the Edit window. If the currently displayed file has been modified a warning message will ask the user to save the changes, or press *Load* again.

RESOURCES

For *xedit* the available resources are:

enableBackups (Class **EnableBackups)**

Specifies that, when edits made to an existing file are saved, *xedit* is to copy the original version of that file to <prefix>*file*<suffix> before it saves the changes. The default value for this resource is ‘‘off,’’ stating that no backups should be created.

backupNamePrefix (Class **BackupNamePrefix)**
Specifies a string that is to be prepended to the backup filename. The default is that no string shall be prepended.

backupNameSuffix (Class **BackupNameSuffix)**
Specifies a string that is to be appended to the backup filename. The default is to use ''.BAK'' as the suffix.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xedit*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
Xedit  xedit
   Paned  paned
      Paned  buttons
         Command  quit
         Command  save
         Command  load
         Text  filename
      Label  bc_label
      Text  messageWindow
      Label  labelWindow
      Text  editWindow
```

ENVIRONMENT VARIABLES

DISPLAY
Contains the default host and display number.

XENVIRONMENT
Contains the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

FILES

$OPENNT_ROOT/usr/lib/X11/app-defaults/Xedit
Specifies required resources.

SEE ALSO

X, xrdb, Athena Widget Set

RESTRICTIONS

There is no *undo* function.

COPYRIGHT

Copyright 1988, Digital Equipment Corporation.
Copyright 1989, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHOR

Chris D. Peterson, MIT X Consortium

NAME

xev – print contents of X events

SYNOPSIS

```
xev [-bs {NotUseful,WhenMapped,Always}] [-bw pixels]
    [-display displayname] [-geometry geom] [-id windowid]
    [-name string] [-rv] [-s]
```

DESCRIPTION

Xev creates a window and then asks the X server to send it *events* whenever anything happens to the window (such as being moved, resized, typed in, clicked in, etc.). You can also attach it to an existing window. It is useful for seeing what causes events to occur and to display the information that they contain.

OPTIONS

–bs *{NotUseful,WhenMapped,Always}*
This option specifies what kind of backing store to give the window. The default is NotUseful.

–bw *pixels*
This option specifies the border width for the window.

–display *display*
This option specifies the X server to contact.

–geometry *geom*
This option specifies the size and / or location of the window.

–id *windowid*
This option specifies that the window with the given id should be monitored, instead of creating a new window.

–rv This option specifies that the window should be in reverse video.

–s This option specifies that save-unders should be enabled on the window.

–name *string*
This option specifies the name to assign to the created window.

SEE ALSO

X, *xwininfo*, *xdpyinfo*, Xlib Programmers Manual, X Protocol Specification

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xeyes – a follow the mouse X demo

SYNOPSIS

`xeyes [-`*`option`* `...]`

DESCRIPTION

Xeyes watches what you do and reports to the Boss.

OPTIONS

–backing {*WhenMapped Always NotUseful*}
selects an appropriate level of backing store.

–bg *background*
choose a different color for the background.

–bd *border*
choose a different color for the window border.

–bw *border*
choose a different width for the window border.

–center *center*
choose a different color for the center of the eyes.

–display *display*
specify the display to use; see *X*.

–fg *foreground*
choose a different color for the pupil of the eyes.

–outline *outline*
choose a different color for the outline of the eyes.

–geometry *geometry*
define the initial window geometry; see *X*.

–shape
uses the SHAPE extension to shape the window. This is the default.

+shape
Disables uses the SHAPE extension to shape the window.

SEE ALSO

X, X Toolkit documentation

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Keith Packard, MIT X Consortium
Copied from the NeWS version written (apparently) by Jeremy Huxtable as seen at SIGGRAPH ’88

NAME

xfd – display all the characters in an X font

SYNOPSIS

`xfd [-options ...] -fn fontname`

DESCRIPTION

The *xfd* utility creates a window containing the name of the font being displayed, a row of command buttons, several lines of text for displaying character metrics, and a grid containing one glyph per cell. The characters are shown in increasing order from left to right, top to bottom. The first character displayed at the top left will be character number 0 unless the **–start** option has been supplied in which case the character with the number given in the **–start** option will be used.

The characters are displayed in a grid of boxes, each large enough to hold any single character in the font. Each character glyph is drawn using the PolyText16 request (used by the *Xlib* routine *XDrawString16*( )). If the **–box** option is given, a rectangle will be drawn around each character, showing where an ImageText16 request (used by the *Xlib* routine **XDrawImageString16**) would cause background color to be displayed.

The origin of each glyph is normally set so that the character is drawn in the upper left hand corner of the grid cell. However, if a glyph has a negative left bearing or an unusually large ascent, descent, or right bearing (as is the case with *cursor* font), some character may not appear in their own grid cells. The **–center** option may be used to force all glyphs to be centered in their respective cells.

All the characters in the font may not fit in the window at once. To see the next page of glyphs, press the *Next* button at the top of the window. To see the previous page, press *Prev*. To exit *xfd*, press *Quit*.

Individual character metrics (index, width, bearings, ascent and descent) can be displayed at the top of the window by pressing on the desired character.

The font name displayed at the top of the window is the full name of the font, as determined by the server. See *xlsfonts* for ways to generate lists of fonts, as well as more detailed summaries of their metrics and properties.

OPTIONS

xfd accepts all of the standard toolkit command line options along with the additional options listed below:

–fn *font*

This option specifies the font to be displayed.

–box

This option indicates that a box should be displayed outlining the area that would be filled with background color by an ImageText request.

–center

This option indicates that each glyph should be centered in its grid.

–start *number*
This option specifies the glyph index of the upper left hand corner of the grid. This is used to view characters at arbitrary locations in the font. The default is 0.

–bc *color*
This option specifies the color to be used if ImageText boxes are drawn.

X DEFAULTS
To be written.

SEE ALSO
X, *xlsfonts*, *xrdb*, *xfontsel*

BUGS
The program should skip over pages full of non-existent characters.

COPYRIGHT
Copyright 1989, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR
Jim Fulton, MIT X Consortium; previous program of the same name by Mark Lillibridge, MIT Project Athena.

NAME

xfontsel - point & click interface for selecting X11 font names

SYNOPSIS

```
xfontsel [-toolkitoption ...] [-pattern fontname] [-print]
         [-sample text] [-sample16 text16] [-noscaled]
```

DESCRIPTION

The *xfontsel* application provides a simple way to display the fonts known to your X server, examine samples of each, and retrieve the X Logical Font Description ("XLFD") full name for a font.

If **–pattern** is not specified, all fonts with XLFD 14-part names will be selectable. To work with only a subset of the fonts, specify **–pattern** followed by a partially or fully qualified font name; e.g., ‘‘-pattern *medium*’’ will select that subset of fonts which contain the string ‘‘medium’’ somewhere in their font name. Be careful about escaping wildcard characters in your shell.

The *xfontsel* command takes the following options:

–noscaled
Disable the ability to select scaled fonts at arbitrary pixel or point sizes. This makes it clear which bitmap sizes are advertised by the server, and can avoid an accidental and sometimes prolonged wait for a font to be scaled.

–print Write the selected font specifier to standard output when the *quit* button is activated. Regardless of whether or not **–print** was specified, the font specifier may be made the PRIMARY (text) selection by activating the *select* button.

–sample *text*
Use *text* to display the selected font if the font is linearly indexed, overriding the default.

–sample16 *text16*
Use *text16* to display the selected font if the font is matrix encoded, overriding the default.

INTERACTIONS

Clicking any pointer button in one of the XLFD field names will pop up a menu of the currently-known possibilities for that field. If previous choices of other fields were made, only values for fonts which matched the previously selected fields will be selectable; to make other values selectable, you must deselect some other field(s) by choosing the ‘‘*’’ entry in that field. Unselectable values may be omitted from the menu entirely as a configuration option; see the **ShowUnselectable** resource, below. Whenever any change is made to a field value, *xfontsel* will assert ownership of the PRIMARY_FONT selection. Other applications (see, e.g., *xterm*) may then retrieve the selected font specification.

Scalable fonts come back from the server with zero for the pixel size, point size, and average width fields. Selecting a font name with a zero in these positions results in an implementation-dependent size. Any pixel or point size can be selected to scale the font to a particular size. Any average width can be

selected to anamorphically scale the font (although you may find this challenging given the size of the average width menu).

Clicking the left pointer button in the *select* widget will cause the currently selected font name to become the PRIMARY text selection as well as the PRIMARY_FONT selection. This then allows you to paste the string into other applications. The **select** button remains highlighted to remind you of this fact, and de-highlights when some other application takes the PRIMARY selection away. The *select* widget is a toggle; pressing it when it is highlighted will cause *xfontsel* to release the selection ownership and de-highlight the widget. Activating the *select* widget twice is the only way to cause *xfontsel* to release the PRIMARY_FONT selection.

RESOURCES

The application class is **XFontSel**. Most of the user-interface is configured in the app-defaults file; if this file is missing a warning message will be printed to standard output and the resulting window will be nearly incomprehensible.

Most of the significant parts of the widget hierarchy are documented in the app-defaults file (normally **$OPENNT_ROOT/usr/X11R5/lib/app-defaults/XFontSel**).

Application specific resources:

cursor (class **Cursor)**
Specifies the cursor for the application window.

pattern (class **Pattern)**
Specifies the font name pattern for selecting a subset of available fonts. Equivalent to the **–pattern** option. Most useful patterns will contain at least one field delimiter; e.g. ‘‘*-m-*’’ for monospaced fonts.

pixelSizeList (class **PixelSizeList)**
Specifies a list of pixel sizes to add to the pixel size menu, so that scalable fonts can be selected at those pixel sizes. The default pixelSizeList contains 7, 30, 40, 50, and 60.

pointSizeList (class **PointSizeList)**
Specifies a list of point sizes (in units of tenths of points) to add to the point size menu, so that scalable fonts can be selected at those point sizes. The default pointSizeList contains 250, 300, 350, and 400.

printOnQuit (class **PrintOnQuit)**
If *True* the currently selected font name is printed to standard output when the quit button is activated. Equivalent to the **–print** option.

sampleText (class **Text)**
The sample 1-byte text to use for linearly indexed fonts. Each glyph index is a single byte, with newline separating lines.

sampleText16 (class **Text16)**
The sample 2-byte text to use for matrix-encoded fonts. Each glyph index is two bytes, with a 1-byte newline separating lines.

scaledFonts (class **ScaledFonts)**
If *True* then selection of arbitrary pixel and point sizes for scalable fonts is enabled.

Widget specific resources:

showUnselectable (class **ShowUnselectable)**
Specifies, for each field menu, whether or not to show values that are not currently selectable, based upon previous field selections. If shown, the unselectable values are clearly identified as such and do not highlight when the pointer is moved down the menu. The full name of this resource is **fieldN.menu.options.showUnselectable**, class **MenuButton.SimpleMenu.Options.ShowUnselectable**; where N is replaced with the field number (starting with the left-most field numbered 0). The default is True for all but field 11 (average width of characters in font) and False for field 11. If you never want to see unselectable entries, '*menu.options.showUnselectable:False' is a reasonable thing to specify in a resource file.

FILES

$XFILESEARCHPATH/XFontSel

SEE ALSO

xrdb, *xfd*

BUGS

Sufficiently ambiguous patterns can be misinterpreted and lead to an initial selection string which may not correspond to what the user intended and which may cause the initial sample text output to fail to match the proffered string. Selecting any new field value will correct the sample output, though possibly resulting in no matching font.

Should be able to return a FONT for the PRIMARY selection, not just a STRING.

Any change in a field value will cause *xfontsel* to assert ownership of the PRIMARY_FONT selection. Perhaps this should be parameterized.

When running on a slow machine, it is possible for the user to request a field menu before the font names have been completely parsed. An error message indicating a missing menu is printed to stderr but otherwise nothing bad (or good) happens.

The average-width menu is too large to be useful.

COPYRIGHT

Copyright 1989, 1991 by the Massachusetts Institute of Technology
See *X* for a full statement of rights and permissions.

AUTHOR

Ralph R. Swick, Digital Equipment Corporation / MIT Project Athena

NAME

xgc - X graphics demo

SYNOPSIS

xgc **[**-*toolkitoption* ...**]**

DESCRIPTION

The *xgc* program demonstrates various features of the X graphics primitives. Try the buttons, see what they do; we haven't the time to document them, perhaps you do?

OPTIONS

Xgc accepts all of the standard X Toolkit command line options.

X DEFAULTS

This program accepts the usual defaults for toolkit applications.

ENVIRONMENT VARIABLES

DISPLAY
: Contains the default host and display number.

XENVIRONMENT
: Contains the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X

BUGS

This program isn't really finished yet.

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHORS

Dan Schmidt, MIT

NAME

xkill – kill a client by its X resource

SYNOPSIS

`xkill [-all] [-button number] [-display` *`displayname`*`]`
`[-frame] [-id` *`resource`*`]`

DESCRIPTION

Xkill is a utility for forcing the X server to close connections to clients. This program is very dangerous, but is useful for aborting programs that have displayed undesired windows on a user's screen. If no resource identifier is given with **–id**, *xkill* will display a special cursor as a prompt for the user to select a window to be killed. If a pointer button is pressed over a non-root window, the server will close its connection to the client that created the window.

OPTIONS

–all This option indicates that all clients with top-level windows on the screen should be killed. *Xkill* will ask you to select the root window with each of the currently defined buttons to give you several chances to abort. Use of this option is highly discouraged.

–button *number*
This option specifies the number of pointer button that should be used in selecting a window to kill. If the word "any" is specified, any button on the pointer may be used. By default, the first button in the pointer map (which is usually the leftmost button) is used.

–display *displayname*
This option specifies the name of the X server to contact.

–frame
This option indicates that xkill should ignore the standard conventions for finding top-level client windows (which are typically nested inside a window manager window), and simply believe that you want to kill direct children of the root.

–id *resource*
This option specifies the X identifier for the resource whose creator is to be aborted. If no resource is specified, *xkill* will display a special cursor with which you should select a window to be kill.

XDEFAULTS

Button Specifies a specific pointer button number or the word "any" to use when selecting windows.

SEE ALSO

X, *xwininfo*, *XKillClient*() and *XGetPointerMapping*() in the Xlib Programmers Manual, KillClient in the X Protocol Specification

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium
Dana Chee, Bellcore

NAME

xlogo - X Window System logo

SYNOPSIS

`xlogo [-`*`toolkitoption`* `...]`

DESCRIPTION

The *xlogo* program displays the X Window System logo. This program is nothing more than a wrapper around the undocumented Athena *Logo* widget.

OPTIONS

Xlogo accepts all of the standard X Toolkit command line options, as well as the following:

–shape

This option indicates that the logo window should be shaped rather than rectangular.

RESOURCES

The default width and the default height are each 100 pixels. This program uses the *Logo* widget in the Athena widget set. It understands all of the Simple widget resource names and classes as well as:

foreground (class **Foreground)**

Specifies the color for the logo. The default is depends on whether *reverseVideo* is specified. If *reverseVideo* is specified the default is *XtDefaultForeground*, otherwise the default is *XtDefaultBackground*.

shapeWindow (class **ShapeWindow)**

Specifies that the window is shaped to the X logo. The default is False.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xlogo*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XLogo  xlogo
   Logo  xlogo
```

ENVIRONMENT VARIABLES

DISPLAY

Contains the default host and display number.

XENVIRONMENT

Contains the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

FILES

/usr/lib/X11/app-defaults/XLogo

specifies required resources

SEE ALSO

X, *xrdb*

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHORS

Ollie Jones of Apollo Computer and Jim Fulton of the MIT X Consortium wrote the logo graphics routine, based on a graphic design by Danny Chong and Ross Chapman of Apollo Computer.

NAME

xlsatoms - list interned atoms defined on server

SYNOPSIS

`xlsatoms [-options ...]`

DESCRIPTION

Xlsatoms lists the interned atoms. By default, all atoms starting from 1 (the lowest atom value defined by the protocol) are listed until unknown atom is found. If an explicit range is given, *xlsatoms* will try all atoms in the range, regardless of whether or not any are undefined.

OPTIONS

–display *dpy*
This option specifies the X server to which to connect.

–format *string*
This option specifies a *printf*( )-style string used to list each atom <*value,name*> pair, printed in that order (*value* is an **unsigned long** and *name* is a **char ***). *Xlsatoms* will supply a newline at the end of each line. The default is *%ld\t%s*.

–name *string*
This option specifies the name of an atom to list. If the atom does not exist, a message will be printed on the standard error.

–range *[low]-[high]*
This option specifies the range of atom values to check. If *low* is not given, a value of 1 assumed. If *high* is not given, *xlsatoms* will stop at the first undefined atom at or above *low*.

SEE ALSO

X, *Xserver*, *xprop*

ENVIRONMENT VARIABLES

DISPLAY
Contains the default host and display to use.

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xlsclients - list client applications running on a display

SYNOPSIS

`xlsclients [-a] [-display displayname] [-l] [-m maxcmdlen]`

DESCRIPTION

Xlsclients is a utility for listing information about the client applications running on a display. It may be used to generate scripts representing a snapshot of the the user's current session.

OPTIONS

–a List clients on all screens. By default, only those clients on the default screen are listed.

–display *displayname*
Use specified X server.

–l Display long listing, showing the window name, icon name, and class hints in addition to the machine name and command string shown in the default listing.

–m *maxcmdlen*
Set to *maxcmdlen* the maximum number of characters in a command that will be printed out. The default is 10000.

ENVIRONMENT VARIABLES

DISPLAY
Contains the default host, display number, and screen.

SEE ALSO

X, *xwininfo*, *xprop*

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xlsfonts - server font list displayer for X

SYNOPSIS

`xlsfonts [-`*`options ...`*`] [-fn `*`pattern`*`]`

DESCRIPTION

Xlsfonts lists the fonts that match the given *pattern*. The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

OPTIONS

–display *host:dpy*
This option specifies the X server to contact.

–l[**l**[**l**]] Generate medium, long, and very long listings, respectively, for each font.

–m Long listings should also print the minimum and maximum bounds of each font.

–C Use multiple columns. This is the same as **–n**.

–1 Use a single column for listings. This is the same as **–n 1.**

–w *width* Use *width* characters for figuring out how many columns to print. The default is 79.

–n *columns*
Use *columns* columns in displaying the output. By default, it will attempt to fit as many columns of font names into the number of character specified by **–w** *width*.

–u Leave the output unsorted.

–o Do an **OpenFont** (and **QueryFont**, if appropriate) rather than a **ListFonts**. This is useful if **ListFonts** or **ListFontsWithInfo** fail to list a known font (as is the case with some scaled font systems).

SEE ALSO

X, Xserver, xset, xfd

ENVIRONMENT VARIABLES

DISPLAY
Contains the default host and display to use.

BUGS

Doing ‘‘xlsfonts -l’’ can tie up your server for a very long time. This is really a bug with single-threaded non-preemptable servers, not with this program.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena; Jim Fulton, MIT X Consortium; Phil Karlton, SGI

NAME

xmag – magnify parts of the screen

SYNOPSIS

`xmag [-mag magfactor] [-source geom] [-toolkitoption ...]`

DESCRIPTION

The *xmag* program allows you to magnify portions of an X screen. If no explicit region is specified, a square with the pointer in the upper left corner is displayed indicating the area to be enlarged. The area can be dragged out to the desired size by pressing Button 2. Once a region has been selected, a window is popped up showing a blown up version of the region in which each pixel in the source image is represented by a small square of the same color. Pressing Button1 in the enlargement window shows the position and RGB value of the pixel under the pointer until the button is released. Typing ''Q'' or ''^C'' in the enlargement window exits the program. The application has 5 buttons across its top.

Close Delete this particular magnification instance.
Replace Bring up the rubber band selector again to select another region for this magnification instance.
New Bring up the rubber band selector to create a new magnification instance.
Cut Put the magnification image into the primary selection.
Paste Copy the primary selection buffer into *xmag*.

Note that you can cut and paste between *xmag* and the *bitmap* program. Resizing *xmag* resizes the magnification area. *xmag* preserves the colormap, visual, and window depth of the source.

WIDGETS

xmag uses the X Toolkit and the Athena Widget Set. The magnified image is displayed in the Scale widget. For more information, see the Athena Widget Set documentation. Below is the widget structure of the *xmag* application. Indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
Xmag xmag
   RootWindow root
   TopLevelShell xmag
      Paned pane1
         Paned pane2
            Command close
            Command replace
            Command new
            Command select
            Command paste
            Label xmag label
         Paned pane2
            Scale scale
   OverrideShell pixShell
      Label pixLabel
```

OPTIONS

–mag *integer*
Use specified magnification. 5 is the default.

–source *geom*
Magnify the size or location specified by *geom*. By default, a 64x64 square is provided for the user to select an area of the screen.

COPYRIGHT

Copyright 1991, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHORS

Dave Sternlicht and Davor Matic, MIT X Consortium.

NAME

xman – Manual page display program for the X Window System

SYNOPSIS

`xman [-`*`options`* `. . . ]`

DESCRIPTION

Xman is a manual page browser. The default size of the initial *xman* window is small so that you can leave it running throughout your entire login session. In the initial window there are three options: *Help* will pop up a window with on-line help, *Quit* will exit, and *Manual Page* will pop up a window with a manual page browser in it. Typing Control-S will pop up a window prompting for a specific manual page to display. You may display more than one manual page browser window at a time from a single execution of *xman*.

For further information on using *xman*, please read the on-line help information. Most of this manual will discuss customization of *xman*.

OPTIONS

Xman supports all standard Toolkit command line arguments (see *X*). The following additional arguments are supported.

–bothshown
Show both the manual page and manual directory to be on the screen at the same time.

–geometry *WxH+X+Y*
Set the size and location of the Top Menu with the three buttons in it.

–helpfile *filename*
Specify a helpfile to use other than the default.

–notopbox
Start without the Top Menu with the three buttons in it.

–pagesize *WxH+X+Y*
Set the size and location of all the Manual Pages.

CUSTOMIZING XMAN

Xman allows customization of both the directories to be searched for manual pages, and the name that each directory will map to in the *Sections* menu. Xman determines which directories it will search by reading the *MANPATH* environment variable. If no *MANPATH* is found then the directory is **/usr/man** is searched on POSIX systems. This environment is expected to be a colon-separated list of directories for *xman* to search.

```
setenv MANPATH /mit/kit/man:/usr/man
```

By default, *xman* will search each of the following directories (in each of the directories specified in the user's *MANPATH*) for manual pages. If manual pages exist in that directory then they are added to list of manual pages for the corresponding menu item. A menu item is only displayed for those sections that actually contain manual pages.

```
Directory    Section Name
---------    ------------
man1   (1) User Commands
man2   (2) System Calls
man3   (3) Subroutines
man4   (4) Devices
man5   (5) File Formats
man6   (6) Games
man7   (7) Miscellaneous
man8   (8) Sys. Administration
manl   (l) Local
mann   (n) New
mano   (o) Old
```

For instance, a user has three directories in her manual path and each contain a directory called **man3**. All these manual pages will appear alphabetically sorted when the user selects the menu item called *(3) Subroutines.* If there is no directory called *mano* in any of the directories in her *MANPATH*, or there are no manual pages in any of the directories called *mano* then no menu item will be displayed for the section called *(o) Old*.

THE MANDESC FILE

By using the **mandesc** file a user or system manager is able to more closely control which manual pages will appear in each of the sections represented by menu items in the *Sections* menu. This functionality is only available on a section by section basis, and individual manual pages may not be handled in this manner.

The format of the **mandesc** file is a character followed by a label. The character determines which of the sections will be added under this label. For instance suppose that you would like to create an extra menu item that contains all programmer subroutines. This label should contain all manual pages in both sections two and three. The **mandesc** file would look like this:

```
2Programmer Subroutines
3Programmer Subroutines
```

This will add a menu item to the *Sections* menu that would bring up a listing of all manual pages in sections two and three of the Programmers Manual. Since the label names are *exactly* the same they will be added to the same section. Note, however, that the original sections still exist.

If you want to completely ignore the default sections in a manual directory then add the line:

```
no default sections
```

anywhere in your **mandesc** file. This keeps *xman* from searching the default manual sections *In that directory only*. As an example, suppose you want to do the same thing as above, but you don't think that it is useful to have the *System Calls* or *Subroutines* sections any longer. You would need to duplicate the default entries, as well as adding your new one.

```
no default sections
1(1) User Commands
2Programmer Subroutines
3Programmer Subroutines
4(4) Devices
5(5) File Formats
6(6) Games
7(7) Miscellaneous
8(8) Sys. Administration
l(l) Local
n(n) New
o(o) Old
```

Xman will read any section that is of the form *man<character>*, where <character> is an upper or lower case letter (they are treated distinctly) or a numeral (0-9). Be warned, however, that man(1) and catman(8) will not search directories that are non-standard.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xman*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
Xman xman    (This widget is never used)
   TopLevelShell  topbox
      Form  form
         Label  topLabel
         Command  helpButton
         Command  quitButton
         Command  manpageButton
      TransientShell  search
         DialogWidgetClass  dialog
            Label  label
            Text  value
            Command  manualPage
            Command  apropos
            Command  cancel
      TransientShell  pleaseStandBy
         Label  label
   TopLevelShell  manualBrowser
      Paned  Manpage_Vpane
         Paned  horizPane
            MenuButton  options
            MenuButton  sections
            Label  manualBrowser
         Viewport  directory
            List  directory
            .
            . (one for each section,
            .  created on the fly)
            .
         ScrollByLine  manualPage
```

```
      SimpleMenu  optionMenu
         SmeBSB  displayDirectory
         SmeBSB  displayManualPage
         SmeBSB  help
         SmeBSB  search
         SmeBSB  showBothScreens
         SmeBSB  removeThisManpage
         SmeBSB  openNewManpage
         SmeBSB  showVersion
         SmeBSB  quit
      SimpleMenu  sectionMenu
         SmeBSB  <name of section>
            .
            . (one for each section)
            .
      TransientShell  search
         DialogWidgetClass  dialog
            Label  label
            Text  value
            Command  manualPage
            Command  apropos
            Command  cancel
      TransientShell  pleaseStandBy
         Label  label
      TransientShell  likeToSave
         Dialog  dialog
            Label  label
            Text  value
            Command  yes
            Command  no
   TopLevelShell  help
      Paned  Manpage_Vpane
         Paned  horizPane
            MenuButton  options
            MenuButton  sections
            Label  manualBrowser
         ScrollByLine  manualPage
      SimpleMenu  optionMenu
         SmeBSB  displayDirectory
         SmeBSB  displayManualPage
         SmeBSB  help
         SmeBSB  search
         SmeBSB  showBothScreens
         SmeBSB  removeThisManpage
         SmeBSB  openNewManpage
         SmeBSB  showVersion
         SmeBSB  quit
```

APPLICATION RESOURCES

xman has the following application-specific resources which allow customizations unique to *xman*.

manualFontNormal (Class **Font**)
The font to use for normal text in the manual pages.

manualFontBold (Class **Font**)
The font to use for bold text in the manual pages.

manualFontItalic (Class **Font**)
The font to use for italic text in the manual pages.

directoryFontNormal (Class **Font**)
The font to use for the directory text.

bothShown (Class **Boolean**)
Either 'true' or 'false,' specifies whether or not you want both the directory and the manual page shown at start up.

directoryHeight (Class **DirectoryHeight**)
The height in pixels of the directory, when the directory and the manual page are shown simultaneously.

topCursor (Class **Cursor**)
The cursor to use in the top box.

helpCursor (Class **Cursor**)
The cursor to use in the help window.

manpageCursor (Class **Cursor**)
The cursor to use in the manual page window.

searchEntryCursor (Class **Cursor**)
The cursor to use in the search entry text widget.

pointerColor (Class **Foreground**)
This is the color of all the cursors (pointers) specified above. The name was chosen to be compatible with xterm.

helpFile (Class **File**)
Use this rather than the system default helpfile.

topBox (Class **Boolean**)
Either 'true' or 'false,' determines whether the top box (containing the help, quit and manual page buttons) or a manual page is put on the screen at start-up. The default is true.

verticalList (Class **Boolean**)
Either 'true' or 'false,' determines whether the directory listing is vertically or horizontally organized. The default is horizontal (false).

GLOBAL ACTIONS

Xman defines all user interaction through global actions. This allows the user to modify the translation table of any widget, and bind any event to the new user action. The list of actions supported by *xman* are:

GotoPage(*page***)**
When used in a manual page display window this will allow the user to move between a directory and manual page display. The *page* argument can be either **Directory** or **ManualPage**.

Quit() This action may be used anywhere, and will exit *xman*.

Search(*type***,** *action***)**
Only useful when used in a search popup, this action will cause the search widget to perform the named search type on the string in the search popup's value widget. This action will also pop down the search widget. The *type* argument can be either **Apropos**, **Manpage** or **Cancel**. If an *action* of **Open** is specified then *xman* will open a new manual page to display the results of the search, otherwise *xman* will attempt to display the results in the parent of the search popup.

PopupHelp()
This action may be used anywhere, and will popup the help widget.

PopupSearch()
This action may be used anywhere except in a help window. It will cause the search popup to become active and visible on the screen, allowing the user search for a manual page.

CreateNewManpage()
This action may be used anywhere, and will create a new manual page display window.

RemoveThisManpage()
This action may be used in any manual page or help display window. When called it will remove the window, and clean up all resources associated with it.

SaveFormattedPage(*action***)**
This action can only be used in the **likeToSave** popup widget, and tells *xman* whether to **Save** or **Cancel** a save of the manual page that has just been formatted.

ShowVersion()
This action may be called from any manual page or help display window, and will cause the informational display line to show the current version of *xman*.

FILES

<*manpath directory*>**/man<***character***>**
<*manpath directory*>**/cat<***character***>**
<*manpath directory*>**/mandesc**
$OPENNT_ROOT/usr/lib/X11/app-defaults/Xman
specifies required resources
$OPENNT_ROOT/tmp
Xman creates temporary files in **$OPENNT_ROOT/tmp** for all unformatted man pages and all apropos searches.

SEE ALSO

X, *man*

catman, *Athena Widget Set*

ENVIRONMENT VARIABLES

DISPLAY
Contains the default host and display to use.

MANPATH
Contains the search path for manual pages. Directories are separated by colons (e.g. /usr/man:/mit/kit/man:/foo/bar/man).

XENVIRONMENT
Contains the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

XAPPLRESDIR
A string that will have ''Xman'' appended to it. This string will be the full path name of a user app-defaults file to be merged into the resource database after the system app-defaults file, and before the resources that are attached to the display.

COPYRIGHT

Copyright 1988 by Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHORS

Chris Peterson, MIT X Consortium from the V10 version written by Barry Shein formerly of Boston University.

NAME

xmbind – Configures virtual key bindings

SYNOPSIS

`xmbind [`*`options`*`] [`*`file`*`]`

DESCRIPTION

xmbind is an X Window System client that configures the virtual key bindings for Motif applications. This action is performed by *mwm* at its startup, so the *xmbind* client is only needed when *mwm* is not in use, or when you want to change bindings without restarting *mwm*. If a file is specified, it's contents are used as the virtual key bindings. If a file is not specified, the file **.motifbind** in the user's home directory is used. If this file is not found, *xmbind* loads the default virtual key bindings, as described in *VirtualBindings*( ).

Options

–display This option specifies the display to use; see X(1).

RELATED INFORMATION

VirtualBindings(1X), *VirtualBindings*(3X).

NAME

xmkmf – create a Makefile from an Imakefile

SYNOPSIS

`xmkmf [-a] [`*`topdir`* `[`*`curdir`*`] ]`

DESCRIPTION

The *xmkmf* command is the normal way to create a **Makefile** from an **Imakefile** shipped with third-party software.

When invoked with no arguments in a directory containing an **Imakefile,** the *imake* program is run with arguments appropriate for your system (configured into *xmkmf* when X was built) and generates a **Makefile**.

When invoked with the **–a** option, *xmkmf* builds the **Makefile** in the current directory, and then automatically executes ‘‘make Makefiles’’ (in case there are subdirectories), ‘‘make includes’’, and ‘‘make depend’’ for you. This is the normal way to configure software that is outside the MIT X build tree.

If working inside the MIT X build tree (unlikely unless you are an X developer, and even then this option is never really used), the *topdir* argument should be specified as the relative pathname from the current directory to the top of the build tree. Optionally, *curdir* may be specified as a relative pathname from the top of the build tree to the current directory. It is necessary to supply *curdir* if the current directory has subdirectories, or the **Makefile** will not be able to build the subdirectories. If a *topdir* is given, *xmkmf* assumes nothing is installed on your system and looks for files in the build tree instead of using the installed versions.

SEE ALSO

imake

NAME

xmodmap - utility for modifying keymaps in X

SYNOPSIS

xmodmap **[**-*options ...*] [*filename*]

DESCRIPTION

The *xmodmap* program is used to edit and display the keyboard *modifier map* and *keymap table* that are used by client applications to convert event keycodes into keysyms. It is usually run from the user's session startup script to configure the keyboard according to personal tastes.

OPTIONS

The following options may be used with *xmodmap*:

–display *display*
Use the specified host and display.

–e *expression*
This option specifies an expression to be executed. Any number of expressions may be specified from the command line.

–grammar
Print on the standard error a help message describing the expression grammar used in files and with expressions.

–help
Print on the standard error a brief description of the command line arguments. This decription is printed whenever you give *xmodmap* an unhandled argument.

–n Do not change the mappings but display what would happen, like *make* does when given this option.

–quiet
Turn off the verbose logging. This is the default.

–pm Print the current modifier map on the standard output.

–pk Print the current keymap table on the standard output.

–pke Print on standard output the current keymap table, in the form of expressions that can be fed back to *xmodmap*.

–pp Print on the standard output the current pointer map.

–verbose
Print logging information as it parses its input.

– A lone dash means that the standard input should be used as the input file.

The *filename* specifies a file containing *xmodmap* expressions to be executed. This file is usually kept in the user's home directory with a name like **.xmodmaprc**.

EXPRESSION GRAMMAR

The *xmodmap* program reads a list of expressions and parses them all before attempting to execute any of them. This makes it possible to refer to keysyms that are being redefined in a natural way without having to worry as much about name conflicts.

keycode *NUMBER* **=** *KEYSYMNAME* ...
The list of keysyms is assigned to the indicated keycode (which may be specified in decimal, hex or octal and can be determined by running the *xev* program in the examples directory).

keysym *KEYSYMNAME* **=** *KEYSYMNAME* ...
The *KEYSYMNAME* on the left hand side is translated into matching keycodes used to perform the corresponding set of **keycode** expressions. The list of keysym names may be found in the header file *<X11/keysymdef.h>* (without the *XK_* prefix) or the keysym database */usr/lib/X11/XKeysymDB*. Note that if the same keysym is bound to multiple keys, the expression is executed for each matching keycode.

clear *MODIFIERNAME*
This removes all entries in the modifier map for the given modifier, where valid name are: **Shift**, **Lock**, **Control**, **Mod1**, **Mod2**, **Mod3**, **Mod4**, and **Mod5** (case does not matter in modifier names, although it does matter for all other names). For example, ''clear Lock'' will remove all any keys that were bound to the shift lock modifier.

add *MODIFIERNAME* **=** *KEYSYMNAME* ...
This adds all keys containing the given keysyms to the indicated modifier map. The keysym names are evaluated after all input expressions are read to make it easy to write expressions to swap keys (see the EXAMPLES section).

remove *MODIFIERNAME* **=** *KEYSYMNAME* ...
This removes all keys containing the given keysyms from the indicated modifier map. Unlike **add,** the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether or not they have been reassigned.

pointer = default
This sets the pointer map back to its default settings (button 1 generates a code of 1, button 2 generates a 2, etc.).

pointer = *NUMBER* **...**
This sets to pointer map to contain the indicated button codes. The list always starts with the first physical button.

Lines that begin with an exclamation point (!) are taken as comments.

If you want to change the binding of a modifier key, you must also remove it from the appropriate modifier map.

EXAMPLES

Many pointers are designed such that the first button is pressed using the index finger of the right hand. People who are left-handed frequently find that it is more comfortable to reverse the button codes that get generated so that the primary button is pressed using the index finger of the left hand. This could be

done on a 3 button pointer as follows:

```
$  xmodmap -e "pointer = 3 2 1"
```

Many editor applications support the notion of Meta keys (similar to Control keys except that Meta is held down instead of Control). However, some servers do not have a Meta keysym in the default keymap table, so one needs to be added by hand. The following command will attach Meta to the Multi-language key (sometimes labeled Compose Character). It also takes advantage of the fact that applications that need a Meta key simply need to get the keycode and don't require the keysym to be in the first column of the keymap table. This means that applications that are looking for a Multi_key (including the default modifier map) won't notice any change.

```
$  xmodmap -e "keysym Multi_key = Multi_key Meta_L"
```

One of the more simple, yet convenient, uses of *xmodmap* is to set the keyboard's "rubout" key to generate an alternate keysym. This frequently involves exchanging Backspace with Delete to be more comfortable to the user. If the *ttyModes* resource in *xterm* is set as well, all terminal emulator windows will use the same key for erasing characters:

```
$  xmodmap -e "keysym BackSpace = Delete"
$  echo "XTerm*ttyModes:  erase ^?" | xrdb -merge
```

Some keyboards do not automatically generate less than and greater than characters when the comma and period keys are shifted. This can be remedied with *xmodmap* by resetting the bindings for the comma and period with the following scripts:

```
!
! make shift-, be < and shift-. be >
!
keysym comma = comma less
keysym period = period greater
```

One of the more irritating differences between keyboards is the location of the Control and Shift Lock keys. A common use of *xmodmap* is to swap these two keys as follows:

```
!
! Swap Caps_Lock and Control_L
!
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

The *keycode* command is useful for assigning the same keysym to multiple keycodes. Although unportable, it also makes it possible to write scripts that can reset the keyboard to a known state. The following script sets the backspace key to generate Delete (as shown above), flushes all existing caps lock bindings, makes the CapsLock key be a control key, make F5 generate Escape, and makes Break / Reset be a shift lock.

```
!
! On the HP, the following keycodes have key caps as listed:
!
!     101  Backspace
!      55  Caps
!      14  Ctrl
!      15  Break/Reset
!      86  Stop
!      89  F5
!
keycode 101 = Delete
keycode 55 = Control_R
clear Lock
add Control = Control_R
keycode 89 = Escape
keycode 15 = Caps_Lock
add Lock = Caps_Lock
```

ENVIRONMENT VARIABLES

DISPLAY
Contains default host and display number.

SEE ALSO

X, *xev*, *Xlib* documentation on key and pointer events

BUGS

Every time a **keycode** expression is evaluated, the server generates a *MappingNotify* event on every client. This can cause some thrashing. All of the changes should be batched together and done at once. Clients that receive keyboard input and ignore *MappingNotify* events will not notice any changes made to keyboard mappings.

Xmodmap should generate "add" and "remove" expressions automatically whenever a keycode that is already bound to a modifier is changed.

There should be a way to have the *remove* expression accept keycodes as well as keysyms for those times when you really mess up your mappings.

COPYRIGHT

Copyright 1988, 1989, 1990 Massachusetts Institute of Technology.
Copyright 1987 Sun Microsystems, Inc.
See *X* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium, rewritten from an earlier version by David Rosenthal of Sun Microsystems.

NAME

xprop - property displayer for X

SYNOPSIS

```
xprop [-display display] [-f atom format [dformat]]*
      [-font font] [-frame] [-fs file] [-grammar]
      [-help] [-id id] [-len n] [-name name]
      [-notype] [-remove property-name] [-root] [-spy]
      [format [dformat] atom]*
```

DESCRIPTION

The *prop* utility is for displaying window and font properties in an X server. One window or font is selected using the command line arguments or possibly in the case of a window, by clicking on the desired window. A list of properties is then given, possibly with formatting information.

OPTIONS

–display *display*
This argument allows you to specify the server to connect to; see *X*.

–f *name format* [*dformat*]
Specifies that the *format* for *name* should be *format* and that the *dformat* for *name* should be *dformat*. If *dformat* is missing, " = $0+ \ n" is assumed.

–font *font*
This argument allows the user to specify that the properties of font *font* should be displayed.

–frame
Specifies that when selecting a window by hand (i.e. if none of **–name**, **–root**, or **–id** are given), look at the window manager frame (if any) instead of looking for the client window.

–fs *file*
Specifies that file *file* should be used as a source of more formats for properties.

–grammar
Print out a detailed grammar for all command line options.

–help Print out a summary of command line options.

–id *id* This argument allows the user to select window *id* on the command line rather than using the pointer to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the pointer might be impossible or interfere with the application.

–len *n* Specifies that at most *n* bytes of any property should be read or displayed.

–name *name*
This argument allows the user to specify that the window named *name* is the target window on the command line rather than using the pointer to select the target window.

–notype
Specifies that the type of each property should not be displayed.

–remove *property-name*
Specifies the name of a property to be removed from the indicated window.

–root This argument specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

–spy Examine window properties forever, looking for property change events.

DESCRIPTION

For each of these properties, its value on the selected window or font is printed using the supplied formatting information if any. If no formatting information is supplied, internal defaults are used. If a property is not defined on the selected window or font, "not defined" is printed as the value for that property. If no property list is given, all the properties possessed by the selected window or font are printed.

A window may be selected in one of four ways. First, if the desired window is the root window, the **–root** argument may be used. If the desired window is not the root window, it may be selected in two ways on the command line, either by id number such as might be obtained from *xwininfo*, or by name if the window possesses a name. The **–id** argument selects a window by id number in either decimal or hex (must start with 0x) while the **–name** argument selects a window by name.

The last way to select a window does not involve the command line at all. If none of **–font**, **–id**, **–name**, and **–root** are specified, a crosshairs cursor is displayed and the user is allowed to choose any visible window by pressing any pointer button in the desired window. If it is desired to display properties of a font as opposed to a window, the **–font** argument must be used.

Other than the above four arguments and the **–help** argument for obtaining help, and the **–grammar** argument for listing the full grammar for the command line, all the other command line arguments are used in specifying both the format of the properties to be displayed and how to display them. The **–len** *n* argument specifies that at most *n* bytes of any given property will be read and displayed. This is useful for example when displaying the cut buffer on the root window which could run to several pages if displayed in full.

Normally each property name is displayed by printing first the property name then its type (if it has one) in parentheses followed by its value. The **–notype** argument specifies that property types should not be displayed. The **–fs** argument is used to specify a file containing a list of formats for properties while the **–f** argument is used to specify the format for one property.

The formatting information for a property actually consists of two parts, a *format* and a *dformat*. The *format* specifies the actual formatting of the property (i.e., is it made up of words, bytes, or longs?, etc.) while the *dformat* specifies how the property should be displayed.

The following paragraphs describe how to construct *format*s and *dformats*. However, for the vast majority of users and uses, this should not be necessary as the built in defaults contain the *format*s and *dformats* necessary to display all the standard properties. It should only be necessary to specify *format*s and *dformats* if a new property is being dealt with or the user dislikes the standard display format. New users especially are encouraged to skip this part.

A *format* consists of one of 0, 8, 16, or 32 followed by a sequence of one or more format characters. The 0, 8, 16, or 32 specifies how many bits per field there are in the property. Zero is a special case meaning use the field size information associated with the property itself. (This is only needed for special cases like type INTEGER which is actually three different types depending on the size of the fields of the property)

A value of 8 means that the property is a sequence of bytes while a value of 16 would mean that the property is a sequence of words. The difference between these two lies in the fact that the sequence of words will be byte swapped while the sequence of bytes will not be when read by a machine of the opposite byte order of the machine that originally wrote the property. For more information on how properties are formatted and stored, consult the Xlib manual.

Once the size of the fields has been specified, it is necessary to specify the type of each field (i.e., is it an integer, a string, an atom, or what?) This is done using one format character per field. If there are more fields in the property than format characters supplied, the last character will be repeated as many times as necessary for the extra fields. The format characters and their meaning are as follows:

a The field holds an atom number. A field of this type should be of size 32.
b The field is an boolean. A 0 means false while anything else means true.
c The field is an unsigned number, a cardinal.
i The field is a signed integer.
m The field is a set of bit flags, 1 meaning on.
s This field and the next ones until either a 0 or the end of the property represent a sequence of bytes. This format character is only usable with a field size of 8 and is most often used to represent a string.
x The field is a hex number (like 'c' but displayed in hex — most useful for displaying window ids and the like)

An example format is "32ica" which is the format for a property of three fields of 32 bits each, the first holding a signed integer, the second an unsigned integer, and the third an atom.

The format of a *dformat*, unlike that of a *format*, is not so rigid. The only limitations on a *dformat* is that one may not start with a letter or a dash. This is so that it can be distinguished from a property name or an argument. A *dformat* is a text string containing special characters instructing that various fields be printed at various points in a manner similar to the formatting string used by

printf. For example, the *dformat* " is ($0, $1 \)\n" would render the POINT 3, -4 which has a *format* of 32ii as " is (3, -4)\n".

Any character other than a $, ?, \, or a (in a *dformat* prints as itself. To print out one of $, ?, \, or (precede it by a \. For example, to print out a $, use \$. Several special backslash sequences are provided as shortcuts. \n will cause a newline to be displayed while \t will cause a tab to be displayed. *o* where *o* is an octal number will display character number *o*.

A $ followed by a number *n* causes field number *n* to be displayed. The format of the displayed field depends on the formatting character used to describe it in the corresponding *format*. I.e., if a cardinal is described by 'c' it will print in decimal while if it is described by a 'x' it is displayed in hex.

If the field is not present in the property (this is possible with some properties), <field not available> is displayed instead. $*n*+ will display field number *n* then a comma then field number *n*+1 then another comma then ... until the last field defined. If field *n* is not defined, nothing is displayed. This is useful for a property that is a list of values.

A ? is used to start a conditional expression, a kind of if-then statement. ?*exp*(*text*) will display *text* if and only if *exp* evaluates to non-zero. This is useful for two things. First, it allows fields to be displayed if and only if a flag is set. And second, it allows a value such as a state number to be displayed as a name rather than as just a number. The syntax of *exp* is as follows:

```
exp     ::= term | term=exp | !exp
term    ::= n | $n | mn
```

The ! operator is a logical ''not'', changing 0 to 1 and any non-zero value to 0. = is an equality operator. Note that internally all expressions are evaluated as 32 bit numbers so -1 is not equal to 65535. = returns 1 if the two values are equal and 0 if not. *n* represents the constant value *n* while $*n* represents the value of field number *n*. m*n* is 1 if flag number *n* in the first field having format character 'm' in the corresponding *format* is 1, 0 otherwise.

Examples: ?m3(count: $3\n) displays field 3 with a label of count if and only if flag number 3 (count starts at 0!) is on. ?$2=0(True)?!$2=0(False) displays the inverted value of field 2 as a boolean.

In order to display a property, *xpropr* needs both a *format* and a *dformat*. Before *xpropr* uses its default values of a *format* of 32x and a *dformat* of " = { $0+ }\n", it searches several places in an attempt to find more specific formats. First, a search is made using the name of the property. If this fails, a search is made using the type of the property. This allows type STRING to be defined with one set of formats while allowing property WM_NAME which is of type STRING to be defined with a different format. In this way, the display formats for a given type can be overridden for specific properties.

The locations searched are in order: the format if any specified with the property name (as in 8x WM_NAME), the formats defined by **–f** options in last to first order, the contents of the file specified by the **–fs** option if any, the contents of the file specified by the environmental variable XPROPFORMATS if any, and finally *xprop*'s built in file of formats.

The format of the files referred to by the **–fs** argument and the XPROPFORMATS variable is one or more lines of the following form:

```
name format [dformat]
```

Where *name* is either the name of a property or the name of a type, *format* is the *format* to be used with *name* and *dformat* is the *dformat* to be used with *name.* If *dformat* is not present, " = $0+\n" is assumed.

EXAMPLES

To display the name of the root window:

```
xprop -root WM_NAME
```

To display the window manager hints for the clock:

```
xprop -name xclock WM_HINTS
```

To display the start of the cut buffer:

```
xprop -root -len 100 CUT_BUFFER0
```

To display the point size of the fixed font:

```
xprop -font fixed POINT_SIZE
```

To display all the properties of window # 0x200007:

```
xprop -id 0x200007
```

ENVIRONMENT VARIABLES

DISPLAY
: Contains default display.

XPROPFORMATS
: Specifies the name of a file from which additional formats are to be obtained.

SEE ALSO

X, xwininfo

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xrdb – X server resource database utility

SYNOPSIS

`xrdb [-`*`option ...`*`] [`*`filename`*`]`

DESCRIPTION

Xrdb is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0, or the SCREEN_RESOURCES property on the root window of any or all screens, or everything combined. You would normally run this program from your X startup file.

Most X clients use the RESOURCE_MANAGER and SCREEN_RESOURCES properties to get user preferences about color, fonts, and so on for applications. Having this information in the server (where it is available to all clients) instead of on disk, solves the problem in previous versions of X that required you to maintain *defaults* files on every machine that you might use. It also allows for dynamic changing of defaults without editing files.

The RESOURCE_MANAGER property is used for resources that apply to all screens of the display. The SCREEN_RESOURCES property on each screen specifies additional (or overriding) resources to be used for that screen. (When there is only one screen, SCREEN_RESOURCES is normally not used, all resources are just placed in the RESOURCE_MANAGER property.)

The file specified by *filename* (or the contents from standard input if – or no filename is given) is optionally passed through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

BITS_PER_RGB=*num*
: the number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it usually is not related to PLANES.

CLASS=*visualclass*
: one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor. This is the visual class of the root window of the default screen.

COLOR
: defined only if CLASS is one of StaticColor, PseudoColor, TrueColor, or DirectColor.

HEIGHT=*num*
: the height of the default screen in pixels.

SERVERHOST=*hostname*
: the hostname portion of the display to which you are connected.

HOST=*hostname*
: the same as **SERVERHOST**.

CLIENTHOST=*hostname*
: the name of the host on which *xrdb* is running.

PLANES=*num*
the number of bit planes (the depth) of the root window of the default screen.

RELEASE=*num*
the vendor release number for the server. The interpretation of this number will vary depending on VENDOR.

REVISION=*num*
the X protocol minor version supported by this server (currently 0).

VERSION=*num*
the X protocol major version supported by this server (should always be 11).

VENDOR=*vendor*
a string specifying the vendor of the server.

WIDTH=*num*
the width of the default screen in pixels.

X_RESOLUTION=*num*
the x resolution of the default screen in pixels per meter.

Y_RESOLUTION=*num*
the y resolution of the default screen in pixels per meter.

Lines that begin with an exclamation mark (!) are ignored and may be used as comments.

Note that since *xrdb* can read from standard input, it can be used to the change the contents of properties directly from a terminal or from a shell script.

OPTIONS

Xrdb program accepts the following options:

–all
Perform the operation on the screen-independent resource property (RESOURCE_MANAGER), as well as the screen-specific property (SCREEN_RESOURCES) on every screen of the display. For example, when used in conjunction with **–query**, the contents of all properties are output. For **–load** and **–merge**, the input file is processed once for each screen. The resources which occur in common in the output for every screen are collected, and these are applied as the screen-independent resources. The remaining resources are applied for each individual per-screen property. This the default mode of operation.

–backup *string*
Generate a backup file with **–edit**, using *string* as a suffix appended to the filename used with **–edit**.

–cpp *filename*
Use *filename* as the C preprocessor program to be ued. Although *xrdb* was designed to use CPP, any program that acts as a filter and accepts the **–D**, **–I**, and **–U** options may be used.

–D *name*[=*value*]
Define symbols for use with conditionals such as **#ifdef**; this option is passed through to the preprocessor.

–display *display*
Use the X server specified by display;see *X*. It also specifies the screen to use for the **–screen** option, and it specifies the screen from which preprocessor symbols are derived for the **–global** option.

–edit *filename*
Edit the contents of the specified properties into the given file, replacing any values already listed there. This allows you to put changes that you have made to your defaults back into your resource file, preserving any comments or preprocessor lines.

–global
Perform the operation only on the screen-independent RESOURCE_MANAGER property.

–help Print a brief description of the allowable options and parameters to be printed; any unsupported option causes the message to be printed.

–I*directory*
Search in *directory* for files referenced with **#include**; this option is passed through to the preprocessor program.

–load Load the input as the new value of the specified properties, replacing whatever was there (i.e. the old contents are removed). This is the default action.

–merge Instead of replacing the current contents of the specified properties, merge the input with it. This option does a lexicographic sorted merge of the two inputs, which is almost certainly not what you want, but remains for backward compatibility.

–n Display the result of the command without really doing it. Changes to the specified properties (when used with **–load** or **–merge**) or to the resource file (when used with **–edit**) are shown on the standard output but not performed.

–nocpp Do not run the input file through a preprocessor before loading it into properties.

–query Print the current contents of the specified properties onto standard output. Since preprocessor commands in the input resource file are part of the input file, not part of the property, they won't appear in the output from this option. The **–edit** option can be used to merge the contents of properties back into the input resource file without damaging preprocessor commands.

–quiet
Don't display warnings about duplicate entries.

–remove
Remove the specified properties from the server.

–retain
Do not reset the server if *xrdb* is the first client. This should never be necessary under normal conditions, since *xdm* and *xinit* always act as the first client.

–screen
Perform the operation only on the SCREEN_RESOURCES property of the default screen of the display.

–screens
perform the operation only on the SCREEN_RESOURCES property of each screen of the display. For **–load** and **–merge**, the input file is processed for each screen.

–symbols
Print onto the standard output the symbols that are defined for the preprocessor.

–U*name*
Remove any definitions of *name*; this option is passed through to the preprocessor.

FILES
Generalizes **~/.Xdefaults** files.

SEE ALSO
X, Xlib Resource Manager documentation, Xt resource documentation

ENVIRONMENT VARIABLES

DISPLAY
Contains which display to use.

BUGS
The default for no arguments should be to query, not to overwrite, so that it is consistent with other programs.

COPYRIGHT
Copyright 1991, Digital Equipment Corporation and MIT.

AUTHORS
Bob Scheifler, Phil Karlton, rewritten from the original by Jim Gettys

NAME

xrefresh – refresh all or part of an X screen

SYNOPSIS

`xrefresh [-`*`option ...`*`]`

DESCRIPTION

Xrefresh is a simple X program that causes all or part of your screen to be repainted. This is useful when system messages have messed up your screen. *Xrefresh* maps a window on top of the desired area of the screen and then immediately unmaps it, causing refresh events to be sent to all applications. By default, a window with no background is used, causing all applications to repaint ''smoothly.'' However, the various options can be used to indicate that a solid background (of any color) or the root window background should be used instead.

ARGUMENTS

–black Use a black background (in effect, turning off all of the electron guns to the tube). This can be somewhat disorienting as everything goes black for a moment.

–display *display*
This argument allows you to specify the server and screen to refresh; see *X*.

–geometry *WxH+X+Y*
Specifies the portion of the screen to be repainted; see *X(1)*.

–none This is the default. All of the windows simply repaint.

–root Use the root window background.

–solid *color*
Use a solid background of the specified color. Try green.

–white Use a white background. The screen just appears to flash quickly, and then repaint.

X DEFAULTS

The *xrefresh* program uses the routine *XGetDefault*( ) to read defaults, so its resource names are all capitalized.

Black, **White**, **Solid**, **None**, **Root**
Determines what sort of window background to use.

Geometry
Determines the area to refresh. Not very useful.

ENVIRONMENT VARIABLES

DISPLAY
Contains default host and display number.

SEE ALSO

X

BUGS

It should have just one default type for the background.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHORS

Jim Gettys, Digital Equipment Corp., MIT Project Athena

NAME

xset - user preference utility for X

SYNOPSIS

```
xset [-display display]
     [-b] [b on/off] [b [volume [pitch [duration]]]
     [[-]bc] [-c] [c on/off] [c [volume]]
     [[-+]fp[-+=] path[,path[,...]]] [fp default] [fp rehash]
     [[-]led [integer]] [led on/off]
     [m[ouse] [accel_mult[/accel_div] [threshold]]] [m[ouse] default]
     [p pixel color] [[-]r] [r on/off]
     [s [length [period]]] [s blank/noblank]
     [s expose/noexpose] [s on/off] [s default] [q]
```

DESCRIPTION

This program is used to set various user preference options of the display.

OPTIONS

–display *display*
This option specifies the server to use; see *X*.

–b The **b** option controls bell volume, pitch and duration. This option accepts up to three numerical parameters, a preceding dash(-), or a 'on/off' flag. If no parameters are given, or the 'on' flag is used, the system defaults will be used. If the dash or 'off' are given, the bell will be turned off. If only one numerical parameter is given, the bell volume will be set to that value, as a percentage of its maximum. Likewise, the second numerical parameter specifies the bell pitch, in hertz, and the third numerical parameter specifies the duration in milliseconds. Note that not all hardware can vary the bell characteristics. The X server will set the characteristics of the bell as closely as it can to the user's specifications.

bc The **bc** option controls *bug compatibility* mode in the server, if possible; a preceding dash(-) disables the mode, otherwise the mode is enabled. Various pre-R4 clients pass illegal values in some protocol requests, and pre-R4 servers did not correctly generate errors in these cases. Such clients, when run against an R4 server, will terminate abnormally or otherwise fail to operate correctly. Bug compatibility mode explicitly reintroduces certain bugs into the X server, so that many such clients can still be run. This mode should be used with care; new application development should be done with this mode disabled. The server must support the MIT-SUNDRY-NONSTANDARD protocol extension in order for this option to work.

c The **c** option controls key click. This option can take an optional value, a preceding dash(-), or an 'on/off' flag. If no parameter or the 'on' flag is given, the system defaults will be used. If the dash or 'off' flag is used, keyclick will be disabled. If a value from 0 to 100 is given, it is used to indicate volume, as a percentage of the maximum. The X server will set the volume to the nearest value that the hardware can support.

fp= *path,...*

The **fp=** sets the font path to the directories given in the path argument. The directories are interpreted by the server, not by the client, and are server-dependent. Directories that do not contain font databases created by *mkfontdir* will be ignored by the server.

fp default

The **default** argument causes the font path to be reset to the server's default.

fp rehash

The **rehash** argument causes the server to reread the font databases in the current font path. This is generally only used when adding new fonts to a font directory (after running *mkfontdir* to recreate the font database).

–fp

fp–

The **–fp** and **fp-** options remove elements from the current font path. They must be followed by a comma-separated list of directories.

+fp or **fp+**

This **+fp** and **fp+** options prepend and append elements to the current font path, respectively. They must be followed by a comma-separated list of directories.

led

The **led** option controls the keyboard LEDs. This controls the turning on or off of one or all of the LEDs. It accepts an optional integer, a preceding dash(-) or an 'on/off' flag. If no parameter or the 'on' flag is given, all LEDs are turned on. If a preceding dash or the flag 'off' is given, all LEDs are turned off. If a value between 1 and 32 is given, that LED will be turned on or off depending on the existence of a preceding dash. A common LED which can be controlled is the ''Caps Lock'' LED. ''xset led 3'' would turn led #3 on. ''xset -led 3'' would turn it off. The particular LED values may refer to different LEDs on different hardware.

m The **m** option controls the mouse parameters. The parameters for the mouse are 'acceleration' and 'threshold'. The acceleration can be specified as an integer, or as a simple fraction. The mouse, or whatever pointer the machine is connected to, will go 'acceleration' times as fast when it travels more than 'threshold' pixels in a short time. This way, the mouse can be used for precise alignment when it is moved slowly, yet it can be set to travel across the screen in a flick of the wrist when desired. One or both parameters for the **m** option can be omitted, but if only one is given, it will be interpreted as the acceleration. If no parameters or the flag 'default' is used, the system defaults will be set.

p The **p** option controls pixel color values. The parameters are the color map entry number in decimal, and a color specification. The root background colors may be changed on some servers by altering the entries for BlackPixel and WhitePixel. Although these are often 0 and

1, they need not be. Also, a server may choose to allocate those colors privately, in which case an error will be generated. The map entry must not be a read-only color, or an error will result.

r The **r** option controls the autorepeat. If a preceding dash or the 'off' flag is used, autorepeat will be disabled. If no parameters or the 'on' flag is used, autorepeat will be enabled.

s The **s** option lets you set the screen saver parameters. This option accepts up to two numerical parameters, a 'blank/noblank' flag, an 'expose/noexpose' flag, an 'on/off' flag, or the 'default' flag. If no parameters or the 'default' flag is used, the system will be set to its default screen saver characteristics. The 'on/off' flags simply turn the screen saver functions on or off. The 'blank' flag sets the preference to blank the video (if the hardware can do so) rather than display a background pattern, while 'noblank' sets the preference to display a pattern rather than blank the video. The 'expose' flag sets the preference to allow window exposures (the server can freely discard window contents), while 'noexpose' sets the preference to disable screen saver unless the server can regenerate the screens without causing exposure events. The length and period parameters for the screen saver function determines how long the server must be inactive for screen saving to activate, and the period to change the background pattern to avoid burn in. The arguments are specified in seconds. If only one numerical parameter is given, it will be used for the length.

q The **q** option gives you information on the current settings.

These settings will be reset to default values when you log out.

Note that not all X implementations are guaranteed to honor all of these options.

SEE ALSO

X, *Xserver*, *xmodmap*, *xrdb*, *xsetroot*

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Bob Scheifler, MIT Laboratory for Computer Science
David Krikorian, MIT Project Athena (X11 version)

NAME

xsetroot – root window parameter setting utility for X

SYNOPSIS

```
xsetroot [-bg color] [-bitmap filename]
            [-cursor cursorfile maskfile] [-cursor_name cursorname]
            [-def] [-display display] [-fg color]
            [-gray] [-grey] [-help] [-mod x y]
            [-name string] [-rv] [-solid color]
```

DESCRIPTION

The *xsetroot* program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with *xsetroot* until you find a personalized look that you like, then put the *xsetroot* command that produces it into your X startup file. If no options are specified, or if **–def** is specified, the window is reset to its default state. The **–def** option can be specified along with other options and only the non-specified characteristics will be reset to the default state.

Only one of the background color / tiling changing options (**–solid**, **–gray, –grey, –bitmap,** and **–mod**) may be specified at a time.

OPTIONS

The various options are as follows:

–bg *color*
Use *color* as the background color.

–bitmap *filename*
Use the bitmap specified in the file to set the window pattern. You can make your own bitmap files (little pictures) using the *bitmap* program. The entire background will be made up of repeated "tiles" of the bitmap.

–cursor *cursorfile maskfile*
This lets you change the pointer cursor to whatever you want when the pointer cursor is outside of any window. Cursor and mask files are bitmaps (little pictures), and can be made with the *bitmap* program. You probably want the mask file to be all black until you get used to the way masks work.

–cursor_name *cursorname*
This lets you change the pointer cursor to one of the standard cursors from the cursor font. Refer to appendix B of the X protocol for the names (except that the XC_ prefix is elided for this option).

–def Reset unspecified attributes to the default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)

–display *display*
Specifies the server to connect to; see *X*.

–fg *color*
Use *color* as the foreground color. Foreground and background colors are meaningful only in combination with -cursor, -bitmap, or -mod.

–gray Make the entire background gray. (Easier on the eyes.)

–grey Make the entire background grey.

–help Print a usage message and exit.

–mod *x y*
This is used if you want a plaid-like grid pattern on your screen. *x* and *y* are integers ranging from 1 to 16. Try the different combinations. Zero and negative numbers are taken as 1.

–name *string*
Set the name of the root window to *string*. There is no default value. Usually a name is assigned to a window so that the window manager can use a text representation when the window is iconified. This option is unused since you can't iconify the background.

–rv This exchanges the foreground and background colors. Normally the foreground color is black and the background color is white.

–solid *color*
This sets the background of the root window to the specified color. This option is only useful on color servers.

SEE ALSO

X, *xset*, *xrdb*

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xstdcmap – X standard colormap utility

SYNOPSIS

```
xstdcmap [-all] [-best] [-blue] [-default]
         [-delete map] [-display display]
         [-gray] [-green] [-help] [-red] [-verbose]
```

DESCRIPTION

The *xstdcmap* utility can be used to selectively define standard colormap properties. It is intended to be run from a user's X startup script to create standard colormap definitions in order to facilitate sharing of scarce colormap resources among clients. Where at all possible, colormaps are created with read-only allocations.

OPTIONS

The following options may be used with *xstdcmap*:

–all This option indicates that all six standard colormap properties should be defined on each screen of the display. Not all screens will support visuals under which all six standard colormap properties are meaningful. *xstdcmap* will determine the best allocations and visuals for the colormap properties of a screen. Any previously existing standard colormap properties will be replaced.

–best This option indicates that the RGB_BEST_MAP should be defined.

–blue This option indicates that the RGB_BLUE_MAP should be defined.

–default
This option indicates that the RGB_DEFAULT_MAP should be defined.

–delete *map*
This option specifies that a standard colormap property should be removed. *map* may be one of: **default**, **best**, **red**, **green**, **blue**, or **gray.**

–display *display*
This option specifies the host and display to use; see *X*.

–gray This option indicates that the RGB_GRAY_MAP should be defined.

–green This option indicates that the RGB_GREEN_MAP should be defined.

–help This option indicates that a brief description of the command line arguments should be printed on the standard error. This will be done whenever an unhandled argument is given to *xstdcmap.*

–red This option indicates that the RGB_RED_MAP should be defined.

–verbose
This option indicates that *xstdcmap* should print logging information as it parses its input and defines the standard colormap properties.

ENVIRONMENT VARIABLES

DISPLAY
Contains default host and display number.

SEE ALSO

X

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Donna Converse, MIT X Consortium

NAME

xterm – terminal emulator for X

SYNOPSIS

xterm **[**–*toolkitoption* ...**]** **[**–*option* ...**]**

DESCRIPTION

The *xterm* program is a terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), *xterm* will use the facilities to notify programs running in the window whenever it is resized.

The VT102 and Tektronix 4014 terminals each have their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (height / width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the "active" window for receiving keyboard input and terminal output. This is the window that contains the text cursor. The active window can be chosen through escape sequences, the "VT Options" menu in the VT102 window, and the "Tek Options" menu in the 4014 window.

EMULATIONS

The VT102 emulation is fairly complete, but does not support the blinking character attribute nor the double-wide and double-size character sets. *Termcap* entries that work with *xterm* include "xterm," "vt102," "vt100" and "ansi," and *xterm* automatically searches the termcap file in this order for these entries and then sets the *TERM* and the *TERMCAP* environment variables.

Many of the special *xterm* features may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences. (See the *Xterm Control Sequences* document.)

The Tektronix 4014 emulation is also fairly good. Four different font sizes and five different lines types are supported. The Tektronix text and graphics commands are recorded internally by *xterm* and may be written to a file by sending the COPY escape sequence (or through the **Tektronix** menu; see below). The name of the file will be "**COPY***yy***–***MM***–***dd***.***hh***:***mm***:***ss*", where *yy*, *MM*, *dd*, *hh*, *mm* and *ss* are the year, month, day, hour, minute and second when the COPY was performed (the file is created in the directory *xterm* is started in, or the home directory for a login *xterm*).

OTHER FEATURES

Xterm automatically highlights the text cursor when the pointer enters the window (selected) and unhighlights it when the pointer leaves the window (unselected). If the window is the focus window, then the text cursor is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the

window. When activated, the current screen is saved and replaced with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The *termcap* entry for *xterm* allows the visual editor *vi* to switch to the alternate screen for editing and to restore the screen on exit.

In either VT102 or Tektronix mode, there are escape sequences to change the name of the windows. See *Xterm Control Sequences* for details.

OPTIONS

The *xterm* terminal emulator accepts all of the standard X Toolkit command line options as well as the following (if the option begins with a '**+**' instead of a '**–**', the option is restored to its default value):

–help This causes *xterm* to print out a verbose message describing its options.

–132 Normally, the VT102 DECCOLM escape sequence that switches between 80 and 132 column mode is ignored. This option causes the DECCOLM escape sequence to be recognized, and the *xterm* window will resize appropriately.

–ah This option indicates that *xterm* should always highlight the text cursor. By default, *xterm* will display a hollow text cursor whenever the focus is lost or the pointer leaves the window.

+ah This option indicates that *xterm* should do text cursor highlighting based on focus.

–b *number* This option specifies the size of the inner border (the distance between the outer edge of the characters and the window border) in pixels. The default is 2.

–cb Set the *vt100* resource **cutToBeginningOfLine** to FALSE.

+cb Set the *vt100* resource **cutToBeginningOfLine** to TRUE.

–cc *characterclassrange*:*value*[,...]
This sets classes indicated by the given ranges for using in selecting by words. See the section specifying character classes.

–cn This option indicates that newlines should not be cut in line-mode selections.

+cn This option indicates that newlines should be cut in line-mode selections.

–cr *color* This option specifies the color to use for text cursor. The default is to use the same foreground color that is used for text.

–cu This option indicates that *xterm* should work around a bug in the *curses* cursor motion package that causes the *more* program to display lines that are exactly the width of the window and are followed by a line beginning with a tab to be displayed incorrectly (the leading tabs are not displayed).

+cu This option indicates that that *xterm* should not work around the *curses* bug mentioned above.

–e *program* [*arguments* ...]
This option specifies the program (and its command line arguments) to be run in the *xterm* window. It also sets the window title and icon name to be the basename of the program being executed if neither *–T* nor *–n* are given on the command line. **This must be the last option on the command line.**

–fb *font* This option specifies a font to be used when displaying bold text. This font must be the same height and width as the normal font. If only one of the normal or bold fonts is specified, it will be used as the normal font and the bold font will be produced by overstriking this font. The default is to do overstriking of the normal font.

–im Turn on the **useInsertMode** resource.

+im Turn off the **useInsertMode** resource.

–j This option indicates that *xterm* should do jump scrolling. Normally, text is scrolled one line at a time; this option allows *xterm* to move multiple lines at a time so that it doesn't fall as far behind. Its use is strongly recommended since it make *xterm* much faster when scanning through large amounts of text. The VT100 escape sequences for enabling and disabling smooth scroll as well as the "VT Options" menu can be used to turn this feature on or off.

+j This option indicates that *xterm* should not do jump scrolling.

–ls This option indicates that the shell that is started in the *xterm* window be a login shell (i.e. the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile).

+ls This option indicates that the shell that is started should not be a login shell (i.e. it will be a normal "subshell"). An entry is automatically added to the **$OPENNT_ROOT/etc/utmpx** file.

–mb This option indicates that *xterm* should ring a margin bell when the user types near the right end of a line. This option can be turned on and off from the "VT Options" menu.

+mb This option indicates that margin bell should not be rung.

–mc *milliseconds*
This option specifies the maximum time between multi-click selections.

–ms *color*
This option specifies the color to be used for the pointer cursor. The default is to use the foreground color.

–nb *number* This option specifies the number of characters from the right end of a line at which the margin bell, if enabled, will ring. The default is 10.

–rw This option indicates that reverse-wraparound should be allowed. This allows the cursor to back up from the leftmost column of one line to the rightmost column of the previous line. This is very useful for editing long shell command lines and is encouraged. This option can be turned on and off from the "VT Options" menu.

+rw This option indicates that reverse-wraparound should not be allowed.

–aw This option indicates that auto-wraparound should be allowed. This allows the cursor to automatically wrap to the beginning of the next line when when it is at the rightmost position of a line and text is output.

+aw This option indicates that auto-wraparound should not be allowed.

–s This option indicates that *xterm* may scroll asynchronously, meaning that the screen does not have to be kept completely up to date while scrolling. This allows *xterm* to run faster when network latencies are very high and is typically useful when running across a very large internet or many gateways.

+s This option indicates that *xterm* should scroll synchronously.

–sb This option indicates that some number of lines that are scrolled off the top of the window should be saved and that a scrollbar should be displayed so that those lines can be viewed. This option may be turned on and off from the "VT Options" menu.

+sb This option indicates that a scrollbar should not be displayed.

–sf This option indicates that Sun Function Key escape codes should be generated for function keys.

+sf This option indicates that the standard escape codes should be generated for function keys.

–si This option indicates that output to a window should not automatically reposition the screen to the bottom of the scrolling region. This option can be turned on and off from the "VT Options" menu.

+si This option indicates that output to a window should cause it to scroll to the bottom.

–sk This option indicates that pressing a key while using the scrollbar to review previous lines of text should cause the window to be repositioned automatically in the normal position at the bottom of the scroll region.

+sk This option indicates that pressing a key while using the scrollbar should not cause the window to be repositioned.

–sl *number* This option specifies the number of lines to save that have been scrolled off the top of the screen. The default is 64.

–t This option indicates that *xterm* should start in Tektronix mode, rather than in VT102 mode. Switching between the two windows is done using the "Options" menus.

+t This option indicates that *xterm* should start in VT102 mode.

–tm *string* This option specifies a series of terminal setting keywords followed by the characters that should be bound to those functions, similar to the *stty* program. Allowable keywords include: intr, quit, erase, kill, eof, eol, swtch, start, stop, brk, susp, dsusp, rprnt, flush, weras, and lnext. Control characters may be specified as ^char (e.g. ^c or ^u) and ^? may be used to indicate delete.

–tn *name* This option specifies the name of the terminal type to be set in the TERM environment variable. This terminal type must exist in the **termcap** database and should have *li#* and *co#* entries.

–ut This option indicates that *xterm* shouldn't write a record into the the system log file **$OPENNT_ROOT/etc/utmpx**. (This option has no effect on INTERIX.)

+ut This option indicates that *xterm* should write a record into the system log file **$OPENNT_ROOT/etc/utmpx**.

–vb This option indicates that a visual bell is preferred over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed.

+vb This option indicates that a visual bell should not be used.

–wf This option indicates that *xterm* should wait for the window to be mapped the first time before starting the subprocess so that the initial terminal size settings and environment variables are correct. It is the application's responsibility to catch subsequent terminal size changes.

+wf This option indicates that *xterm* show not wait before starting the subprocess.

–C This option indicates that this window should receive console output. This is not supported on all systems. To obtain console output, you must be the owner of the console device, and you must have read and write permission for it. If you are running X under *xdm* on the console screen you may need to have the session startup and reset programs explicitly change the ownership of the console device in order to get this option to work.

–S*ccn* This option specifies the last two letters of the name of a pseudoterminal to use in slave mode, plus the number of the inherited file descriptor. The option is parsed "%c%c%d". This allows *xterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications.

The following command line arguments are provided for compatibility with older versions. They may not be supported in the next release as the X Toolkit provides standard options that accomplish the same task.

%*geom* This option specifies the preferred size and position of the Tektronix window. It is shorthand for specifying the "**tekGeometry*" resource.

#*geom* This option specifies the preferred position of the icon window. It is shorthand for specifying the "**iconGeometry*" resource.

–T *string* This option specifies the title for *xterm*'s windows. It is equivalent to **–title**.

–n *string* This option specifies the icon name for *xterm*'s windows. It is shorthand for specifying the "**iconName*" resource. Note that this is not the same as the toolkit option **–name** (see below). The default icon name is the application name.

–r This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to **–rv**.

–w *number* This option specifies the width in pixels of the border surrounding the window. It is equivalent to **–borderwidth** or **–bw**.

The following standard X Toolkit command line arguments are commonly used with *xterm*:

–bg *color*
This option specifies the color to use for the background of the window. The default is "white."

–bd *color*
This option specifies the color to use for the border of the window. The default is "black."

–bw *number*
This option specifies the width in pixels of the border surrounding the window.

–fg *color*
This option specifies the color to use for displaying text. The default is "black."

–fn *font*
This option specifies the font to be used for displaying normal text. The default is *fixed*.

–name *name*
This option specifies the application name under which resources are to be obtained, rather than the default executable file name. *Name* should not contain "." or "*" characters.

–title *string*
This option specifies the window title string, which may be displayed by window managers if the user so chooses. The default title is the command line specified after the **—e** option, if any, otherwise the application name.

–rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

–geometry *geometry*
This option specifies the preferred size and position of the VT102 window; see *X*.

–display *display*
This option specifies the X server to contact; see *X*.

–xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

–iconic
This option indicates that *xterm* should ask the window manager to start it as an icon rather than as the normal window.

RESOURCES

The program understands all of the core X Toolkit resource names and classes as well as:

iconGeometry (class **IconGeometry)**
Specifies the preferred size and position of the application when iconified. It is not necessarily obeyed by all window managers.

iconName (class **IconName)**
Specifies the icon name. The default is the application name.

termName (class **TermName)**
Specifies the terminal type name to be set in the *TERM* environment variable.

title (class **Title)**
Specifies a string that may be used by the window manager when displaying this application.

ttyModes (class **TtyModes)**
Specifies a string containing terminal setting keywords and the characters to which they may be bound. Allowable keywords include: intr, quit, erase, kill, eof, eol, swtch, start, stop, brk, susp, dsusp, rprnt, flush, weras, and lnext. Control characters may be specified as ^char (e.g. ^c or ^u) and ^? may be used to indicate Delete. This is very useful for overriding the default terminal settings without having to do an *stty* every time an *xterm* is started.

useInsertMode (class **UseInsertMode)**
Force use of insert mode by adding appropriate entries to the TERMCAP environment variable. This is useful if the system termcap is broken. The default is "false."

utmpInhibit (class **UtmpInhibit)**

Specifies whether or not *xterm* should try to record the user's terminal in **$OPENNT_ROOT/etc/utmpx**.

sunFunctionKeys (class **SunFunctionKeys)**
Specifies whether or not Sun Function Key escape codes should be generated for function keys instead of standard escape sequences.

waitForMap (class **WaitForMap)**
Specifies whether or not *xterm* should wait for the initial window map before starting the subprocess. The default is "false."

The following resources are specified as part of the *vt100* widget (class *VT100*):

allowSendEvents (class **AllowSendEvents)**
Specifies whether or not synthetic key and button events (generated using the X protocol SendEvent request) should be interpreted or discarded. The default is "false" meaning they are discarded. Note that allowing such events creates a very large security hole.

"**alwaysHighlight (**class **AlwaysHighlight)**
Specifies whether or not *xterm* should always display a highlighted text cursor. By default, a hollow text cursor is displayed whenever the pointer moves out of the window or the window loses the input focus.

appcursorDefault (class **AppcursorDefault)**
If "true," the cursor keys are initially in application mode. The default is "false."

appkeypadDefault (class **AppkeypadDefault)**
If "true," the keypad keys are initially in application mode. The default is "false."

autoWrap (class **AutoWrap)**
Specifies whether or not auto-wraparound should be enabled. The default is "true."

bellSuppressTime (class **BellSuppressTime)**
Number of milliseconds after a bell command is sent during which additional bells will be suppressed. Default is 200. If set non-zero, additional bells will also be suppressed until the server reports that processing of the first bell has been completed; this feature is most useful with the visible bell.

boldFont (class **BoldFont)**
Specifies the name of the bold font to use instead of overstriking.

c132 (class **C132)**
Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is "false."

cutNewline (class **CutNewline)**
If false, triple clicking to select a line does not include the Newline at the end of the line. If true, the Newline is selected. The default is "true."

cutToBeginningOfLine (class **CutToBeginningOfLine)**
If false, triple clicking to select a line selects only from the current word forward. If true, the entire line is selected. The default is "true."

charClass (class **CharClass)**
Specifies comma-separated lists of character class bindings of the form [*low*-]*high*:*value*. These are used in determining which sets of characters should be treated the same when doing cut and paste. See the section on specifying character classes.

curses (class **Curses)**
Specifies whether or not the last column bug in *curses* should be worked around. The default is “false.”

background (class **Background)**
Specifies the color to use for the background of the window. The default is “white.”

foreground (class **Foreground)**
Specifies the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the text color change color. The default is “black.”

cursorColor (class **Foreground)**
Specifies the color to use for the text cursor. The default is “black.”

eightBitInput (class **EightBitInput**)
If true, Meta characters input from the keyboard are presented as a single character with the eighth bit turned on. If false, Meta characters are converted into a two-character sequence with the character itself preceded by ESC. The default is “true.”

eightBitOutput (class **EightBitOutput**)
Specifies whether or not eight-bit characters sent from the host should be accepted as is or stripped when printed. The default is “true.”

font (class **Font)**
Specifies the name of the normal font. The default is “fixed.”

font1 (class **Font1)**
Specifies the name of the first alternative font.

font2 (class **Font2)**
Specifies the name of the second alternative font.

font3 (class **Font3)**
Specifies the name of the third alternative font.

font4 (class **Font4)**
Specifies the name of the fourth alternative font.

font5 (class **Font5)**
Specifies the name of the fifth alternative font.

font6 (class **Font6)**
Specifies the name of the sixth alternative font.

geometry (class **Geometry)**
Specifies the preferred size and position of the VT102 window.

internalBorder (class **BorderWidth)**
Specifies the number of pixels between the characters and the window border. The default is 2.

jumpScroll (class **JumpScroll)**
Specifies whether or not jump scroll should be used. The default is “true.”

loginShell (class **LoginShell)**
Specifies whether or not the shell to be run in the window should be started as a login shell. The default is “false.”

marginBell (class **MarginBell)**
Specifies whether or not the bell should be run when the user types near the right margin. The default is “false.”

multiClickTime (class **MultiClickTime)**
Specifies the maximum time in milliseconds between multi-click

select events. The default is 250 milliseconds.

multiScroll (class **MultiScroll)**
Specifies whether or not scrolling should be done asynchronously. The default is "false."

nMarginBell (class **Column)**
Specifies the number of characters from the right margin at which the margin bell should be rung, when enabled.

pointerColor (class **Foreground)**
Specifies the foreground color of the pointer. The default is "XtDefaultForeground."

pointerColorBackground (class **Background)**
Specifies the background color of the pointer. The default is "XtDefaultBackground."

pointerShape (class **Cursor)**
Specifies the name of the shape of the pointer. The default is "xterm."

resizeGravity (class **ResizeGravity)**
Affects the behavior when the window is resized to be taller or shorter. **NorthWest** specifies that the top line of text on the screen stay fixed. If the window is made shorter, lines are dropped from the bottom; if the window is made taller, blank lines are added at the bottom. This is compatible with the behavior in R4. **SouthWest** (the default) specifies that the bottom line of text on the screen stay fixed. If the window is made taller, additional saved lines will be scrolled down onto the screen; if the window is made shorter, lines will be scrolled off the top of the screen, and the top saved lines will be dropped.

reverseVideo (class **ReverseVideo)**
Specifies whether or not reverse video should be simulated. The default is "false."

reverseWrap (class **ReverseWrap)**
Specifies whether or not reverse-wraparound should be enabled. The default is "false."

saveLines (class **SaveLines)**
Specifies the number of lines to save beyond the top of the screen when a scrollbar is turned on. The default is 64.

scrollBar (class **ScrollBar)**
Specifies whether or not the scrollbar should be displayed. The default is "false."

scrollTtyOutput (class **ScrollCond)**
Specifies whether or not output to the terminal should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "true."

scrollKey (class **ScrollCond)**
Specifies whether or not pressing a key should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "false."

scrollLines (class **ScrollLines)**
Specifies the number of lines that the *scroll-back* and *scroll-forw* actions should use as a default. The default value is 1.

signalInhibit (class **SignalInhibit)**
Specifies whether or not the entries in the "Main Options" menu for

sending signals to *xterm* should be disallowed. The default is "false."

tekGeometry (class **Geometry)**
Specifies the preferred size and position of the Tektronix window.

tekInhibit (class **TekInhibit)**
Specifies whether or not Tektronix mode should be disallowed. The default is "false."

tekSmall (class **TekSmall)**
Specifies whether or not the Tektronix mode window should start in its smallest size if no explicit geometry is given. This is useful when running *xterm* on displays with small screens. The default is "false."

tekStartup (class **TekStartup)**
Specifies whether or not *xterm* should start up in Tektronix mode. The default is "false."

titeInhibit (class **TiteInhibit)**
Specifies whether or not *xterm* should remove remove **ti** and **te** termcap entries (used to switch between alternate screens on startup of many screen-oriented programs) from the TERMCAP string. If set, *xterm* also ignores the escape sequence to switch to the alternate screen.

translations (class **Translations)**
Specifies the key and button bindings for menus, selections, ''programmed strings,'' etc. See **ACTIONS** below.

visualBell (class **VisualBell)**
Specifies whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received. The default is "false."

The following resources are specified as part of the *tek4014* widget (class *Tek4014*):

width (class **Width)**
Specifies the width of the Tektronix window in pixels.

height (class **Height)**
Specifies the height of the Tektronix window in pixels.

fontLarge (class **Font)**
Specifies the large font to use in the Tektronix window.

font2 (class **Font)**
Specifies font number 2 to use in the Tektronix window.

font3 (class **Font)**
Specifies font number 3 to use in the Tektronix window.

fontSmall (class **Font)**
Specifies the small font to use in the Tektronix window.

initialFont (class **InitialFont)**
Specifies which of the four Tektronix fonts to use initially. Values are the same as for the *set-tek-text* action. The default is "large."

ginTerminator (class **GinTerminator)**
Specifies what character(s) should follow a GIN report or status report. The possibilities are "none," which sends no terminating characters, "CRonly," which sends CR, and "CR&EOT," which sends both CR and EOT. The default is "none."

The resources that may be specified for the various menus are described in the documentation for the Athena **SimpleMenu** widget. The name and classes of

the entries in each of the menus are listed below.

The *mainMenu* has the following entries:

securekbd (class **SmeBSB)**
This entry invokes the **secure()** action.
allowsends (class **SmeBSB)**
This entry invokes the **allow-send-events(toggle)** action.
redraw (class **SmeBSB)**
This entry invokes the **redraw()** action.
line1 (class **SmeLine)**
This is a separator.
suspend (class **SmeBSB)**
This entry invokes the **send-signal(tstp)** action on systems that support job control.
continue (class **SmeBSB)**
This entry invokes the **send-signal(cont)** action on systems that support job control.
interrupt (class **SmeBSB)**
This entry invokes the **send-signal(int)** action.
hangup (class **SmeBSB)**
This entry invokes the **send-signal(hup)** action.
terminate (class **SmeBSB)**
This entry invokes the **send-signal(term)** action.
kill (class **SmeBSB)**
This entry invokes the **send-signal(kill)** action.
line2 (class **SmeLine)**
This is a separator.
quit (class **SmeBSB)**
This entry invokes the **quit()** action.

The *vtMenu* has the following entries:

scrollbar (class **SmeBSB)**
This entry invokes the **set-scrollbar(toggle)** action.
jumpscroll (class **SmeBSB)**
This entry invokes the **set-jumpscroll(toggle)** action.
reversevideo (class **SmeBSB)**
This entry invokes the **set-reverse-video(toggle)** action.
autowrap (class **SmeBSB)**
This entry invokes the **set-autowrap(toggle)** action.
reversewrap (class **SmeBSB)**
This entry invokes the **set-reversewrap(toggle)** action.
autolinefeed (class **SmeBSB)**
This entry invokes the **set-autolinefeed(toggle)** action.
appcursor (class **SmeBSB)**
This entry invokes the **set-appcursor(toggle)** action.
appkeypad (class **SmeBSB)**
This entry invokes the **set-appkeypad(toggle)** action.
scrollkey (class **SmeBSB)**
This entry invokes the **set-scroll-on-key(toggle)** action.
scrollttyoutput (class **SmeBSB)**
This entry invokes the **set-scroll-on-tty-output(toggle)** action.

allow132 (class **SmeBSB)**
This entry invokes the **set-allow132(toggle)** action.
cursesemul (class **SmeBSB)**
This entry invokes the **set-cursesemul(toggle)** action.
visualbell (class **SmeBSB)**
This entry invokes the **set-visualbell(toggle)** action.
marginbell (class **SmeBSB)**
This entry invokes the **set-marginbell(toggle)** action.
altscreen (class **SmeBSB)**
This entry is currently disabled.
line1 (class **SmeLine)**
This is a separator.
softreset (class **SmeBSB)**
This entry invokes the **soft-reset()** action.
hardreset (class **SmeBSB)**
This entry invokes the **hard-reset()** action.
clearsavedlines
This entry invokes the clear-saved-lines() action.
line2 (class SmeLine)
This is a separator.
tekshow (class **SmeBSB)**
This entry invokes the **set-visibility(tek,toggle)** action.
tekmode (class **SmeBSB)**
This entry invokes the **set-terminal-type(tek)** action.
vthide (class **SmeBSB)**
This entry invokes the **set-visibility(vt,off)** action.

The *fontMenu* has the following entries:

fontdefault (class **SmeBSB)**
This entry invokes the **set-vt-font(d)** action.
font1 (class **SmeBSB)**
This entry invokes the **set-vt-font(1)** action.
font2 (class **SmeBSB)**
This entry invokes the **set-vt-font(2)** action.
font3 (class **SmeBSB)**
This entry invokes the **set-vt-font(3)** action.
font4 (class **SmeBSB)**
This entry invokes the **set-vt-font(4)** action.
font5 (class **SmeBSB)**
This entry invokes the **set-vt-font(5)** action.
font6 (class **SmeBSB)**
This entry invokes the **set-vt-font(6)** action.
fontescape (class **SmeBSB)**
This entry invokes the **set-vt-font(e)** action.
fontsel (class **SmeBSB)**
This entry invokes the **set-vt-font(s)** action.

The *tekMenu* has the following entries:

tektextlarge (class **SmeBSB)**
This entry invokes the **set-tek-text(l)** action.
tektext2 (class **SmeBSB)**

This entry invokes the **set-tek-text(2)** action.

tektext3 (class **SmeBSB)**
This entry invokes the **set-tek-text(3)** action.

tektextsmall (class **SmeBSB)**
This entry invokes the **set-tek-text(s)** action.

line1 (class **SmeLine)**
This is a separator.

tekpage (class **SmeBSB)**
This entry invokes the **tek-page()** action.

tekreset (class **SmeBSB)**
This entry invokes the **tek-reset()** action.

tekcopy (class **SmeBSB)**
This entry invokes the **tek-copy()** action.

line2 (class **SmeLine)**
This is a separator.

vtshow (class **SmeBSB)**
This entry invokes the **set-visibility(vt,toggle)** action.

vtmode (class **SmeBSB)**
This entry invokes the **set-terminal-type(vt)** action.

tekhide (class **SmeBSB)**
This entry invokes the **set-visibility(tek,toggle)** action.

The following resources are useful when specified for the Athena Scrollbar widget:

thickness (class **Thickness)**
Specifies the width in pixels of the scrollbar.

background (class **Background)**
Specifies the color to use for the background of the scrollbar.

foreground (class **Foreground)**
Specifies the color to use for the foreground of the scrollbar. The "thumb" of the scrollbar is a simple checkerboard pattern alternating pixels for foreground and background color.

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the "shift" key. The assignment of the functions described below to keys and buttons may be changed through the resource database; see **ACTIONS** below.

Pointer button one (usually left) is used to save text into the cut buffer. Move the cursor to beginning of the text, and then hold the button down while moving the cursor to the end of the region and releasing the button. The selected text is highlighted and is saved in the global cut buffer and made the PRIMARY selection when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection. If the key / button bindings specify that an X selection is to be made, *xterm* will leave the selected text highlighted for as long as it is the selection owner.

Pointer button two (usually middle) 'types' (pastes) the text from the PRIMARY selection, if any, otherwise from the cut buffer, inserting it as keyboard input.

Pointer button three (usually right) extends the current selection. (Without loss of generality, you can swap "right" and "left" everywhere in the rest of this paragraph.) If pressed while closer to the right edge of the selection than the left, it extends/contracts the right edge of the selection. If you contract the selection past the left edge of the selection, *xterm* assumes you really meant the left edge, restores the original selection, then extends/contracts the left edge of the selection. Extension starts in the selection unit mode that the last selection or extension was performed in; you can multiple-click to cycle through them.

By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Since the cut buffer is globally shared among different applications, you should regard it as a 'file' whose contents you know. The terminal emulator and other text programs should be treating it as if it were a text file, i.e., the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

Clicking button one with the pointer in the scroll region moves the adjacent line to the top of the display window.

Clicking button three moves the top line of the display window down to the pointer position.

Clicking button two moves the display to a position in the saved text that corresponds to the pointer's position in the scrollbar.

Unlike the VT102 window, the Tektronix window dows not allow the copying of text. It does allow Tektronix GIN mode, and in this mode the cursor will change from an arrow to a cross. Pressing any key will send that key and the current coordinate of the cross cursor. Pressing button one, two, or three will return the letters 'l', 'm', and 'r', respectively. If the 'shift' key is pressed when a pointer button is pressed, the corresponding upper case letter is sent. To distinguish a pointer button from a key, the high bit of the character is set (but this is bit is normally stripped unless the terminal mode is RAW; see *tty*(4) for details).

MENUS

Xterm has four menus, named *mainMenu*, *vtMenu*, *fontMenu*, and *tekMenu*. Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two section, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the "control" key and pointer button one are pressed in a window. The *mainMenu* contains items that apply to both the VT102 and Tektronix windows. The **Secure**Keyboard mode is be used when typing in passwords or other sensitive data in an unsecure environment; see **SECURITY** below. Notable entries in the command section of the menu are the **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate** and **Kill** which sends the SIGCONT, SIGTSTP, SIGINT, SIGHUP, SIGTERM and SIGKILL signals, respectively, to the process group of the process running under *xterm* (usually the shell). The **Continue** function is especially useful if the user has accidentally typed CTRL-Z, suspending the process.

The *vtMenu* sets various modes in the VT102 emulation, and is popped up when the "control" key and pointer button two are pressed in the VT102 window. In the command section of this menu, the soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes (such as wrap and smooth scroll) to their initial states just after *xterm* has finished processing the command line options.

The *fontMenu* sets the font used in the VT102 window. In addition to the default font and a number of alternatives that are set with resources, the menu offers the font last specified by the Set Font escape sequence (see the document *Xterm Control Sequences*) and the current selection as a font name (if the PRIMARY selection is owned).

The *tekMenu* sets various modes in the Tektronix emulation, and is popped up when the "control" key and pointer button two are pressed in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

SECURITY

X environments differ in their security consciousness. MIT servers, run under *xdm*, are capable of using a "magic cookie" authorization scheme that can provide a reasonable level of security for many people. If your server is only using a host-based mechanism to control access to the server (see *xhost(1)*), then if you enable access for a host and other users are also permitted to run clients on that same host, there is every possibility that someone can run an application that will use the basic services of the X protocol to snoop on your activities, potentially capturing a transcript of everything you type at the keyboard. This is of particular concern when you want to type in a password or other sensitive data. The best solution to this problem is to use a better authorization mechanism that host-based control, but a simple mechanism exists for protecting keyboard input in *xterm*.

The **xterm** menu (see **MENUS** above) contains a **Secure Keyboard** entry which, when enabled, ensures that all keyboard input is directed *only* to *xterm* (using the GrabKeyboard protocol request). When an application prompts you for a password (or other sensitive data), you can enable **Secure Keyboard** using the menu, type in the data, and then disable **Secure Keyboard** using the menu again. Only one X client at a time can secure the keyboard, so when you attempt to enable **Secure Keyboard** it may fail. In this case, the bell will sound. If the **Secure Keyboard** succeeds, the foreground and background colors will

be exchanged (as if you selected the **Reverse Video** entry in the **Modes** menu); they will be exchanged again when you exit secure mode. If the colors do *not* switch, then you should be *very* suspicious that you are being spoofed. If the application you are running displays a prompt before asking for the password, it is safest to enter secure mode *before* the prompt gets displayed, and to make sure that the prompt gets displayed correctly (in the new colors), to minimize the probability of spoofing. You can also bring up the menu again and make sure that a check mark appears next to the entry.

Secure Keyboard mode will be disabled automatically if your xterm window becomes iconified (or otherwise unmapped), or if you start up a reparenting window manager (that places a title bar or other decoration around the window) while in **Secure Keyboard** mode. (This is a feature of the X protocol not easily overcome.) When this happens, the foreground and background colors will be switched back and the bell will sound in warning.

CHARACTER CLASSES

Clicking the middle mouse button twice in rapid succession will cause all characters of the same class (e.g. letters, white space, punctuation) to be selected. Since different people have different preferences for what should be selected (for example, should filenames be selected as a whole or only the separate subnames), the default mapping can be overridden through the use of the *charClass* (class *CharClass*) resource.

This resource is simply a list of *range*:*value* pairs where the range is either a single number or *low*-*high* in the range of 0 to 127, corresponding to the ASCII code for the character or characters to be set. The *value* is arbitrary, although the default table uses the character number of the first character occurring in the set.

The default table is

```
static int charClass[128] = {
/* NUL  SOH  STX  ETX  EOT  ENQ  ACK  BEL */
    32,   1,   1,   1,   1,   1,   1,   1,
/*  BS   HT   NL   VT   NP   CR   SO   SI */
     1,  32,   1,   1,   1,   1,   1,   1,
/* DLE  DC1  DC2  DC3  DC4  NAK  SYN  ETB */
     1,   1,   1,   1,   1,   1,   1,   1,
/* CAN   EM  SUB  ESC   FS   GS   RS   US */
     1,   1,   1,   1,   1,   1,   1,   1,
/*  SP    !    "    #    $    %    &    ' */
    32,  33,  34,  35,  36,  37,  38,  39,
/*   (    )    *    +    ,    -    .    / */
    40,  41,  42,  43,  44,  45,  46,  47,
/*   0    1    2    3    4    5    6    7 */
    48,  48,  48,  48,  48,  48,  48,  48,
/*   8    9    :    ;    <    =    >    ? */
    48,  48,  58,  59,  60,  61,  62,  63,
/*   @    A    B    C    D    E    F    G */
    64,  48,  48,  48,  48,  48,  48,  48,
/*   H    I    J    K    L    M    N    O */
    48,  48,  48,  48,  48,  48,  48,  48,
/*   P    Q    R    S    T    U    V    W */
    48,  48,  48,  48,  48,  48,  48,  48,
/*   X    Y    Z    [         ]    ^    _ */
    48,  48,  48,  91,  92,  93,  94,  48,
/*   `    a    b    c    d    e    f    g */
    96,  48,  48,  48,  48,  48,  48,  48,
/*   h    i    j    k    l    m    n    o */
    48,  48,  48,  48,  48,  48,  48,  48,
/*   p    q    r    s    t    u    v    w */
    48,  48,  48,  48,  48,  48,  48,  48,
/*   x    y    z    {    |    }    ~  DEL */
    48,  48,  48, 123, 124, 125, 126,   1};
```

For example, the string “33:48,37:48,45-47:48,64:48” indicates that the exclamation mark, percent sign, dash, period, slash, and ampersand characters should be treated the same way as characters and numbers. This is very useful for cutting and pasting electronic mailing addresses and filenames.

ACTIONS

It is possible to rebind keys (or sequences of keys) to arbitrary strings for input, by changing the translations for the vt100 or tek4014 widgets. Changing the translations for events other than key and button events is not expected, and will cause unpredictable behavior. The following actions are provided for using within the *vt100* or *tek4014* **translations** resources:

bell([*percent*])
: This action rings the keyboard bell at the specified percentage above or below the base volume.

ignore()
: This action ignores the event but checks for special pointer position escape sequences.

insert() This action inserts the character or string associated with the key that

was pressed.

insert-seven-bit()
This action is a synonym for **insert()**

insert-eight-bit()
This action inserts an eight-bit (Meta) version of the character or string associated with the key that was pressed. The exact action depends on the value of the **eightBitInput** resource.

insert-selection(*sourcename* **[, ...])**
This action inserts the string found in the selection or cutbuffer indicated by *sourcename*. Sources are checked in the order given (case is significant) until one is found. Commonly-used selections include: *PRIMARY*, *SECONDARY*, and *CLIPBOARD*. Cut buffers are typically named *CUT_BUFFER0* through *CUT_BUFFER7*.

keymap(*name*)
This action dynamically defines a new translation table whose resource name is *name* with the suffix *Keymap* (case is significant). The name *None* restores the original translation table.

popup-menu(*menuname*)
This action displays the specified popup menu. Valid names (case is significant) include: *mainMenu*, *vtMenu*, *fontMenu*, and *tekMenu*.

secure()
This action toggles the *Secure Keyboard* mode described in the section named **SECURITY**, and is invoked from the **securekbd** entry in *mainMenu*.

select-start()
This action begins text selection at the current pointer location. See the section on **POINTER USAGE** for information on making selections.

select-extend()
This action tracks the pointer and extends the selection. It should only be bound to Motion events.

select-end(*destname* **[, ...])**
This action puts the currently selected text into all of the selections or cutbuffers specified by *destname*.

select-cursor-start()
This action is similar to **select-start** except that it begins the selection at the current text cursor position.

select-cursor-end(*destname* **[, ...])**
This action is similar to **select-end** except that it should be used with **select-cursor-start**.

set-vt-font(*d/1/2/3/4/5/6/e/s* **[,***normalfont* **[,** *boldfont*]])
This action sets the font or fonts currently being used in the VT102 window. The first argument is a single character that specifies the font to be used: *d* or *D* indicate the default font (the font initially used when *xterm* was started), *1* through *6* indicate the fonts specified by the *font1* through *font6* resources, *e* or *E* indicate the normal and bold fonts that have been set through escape codes (or specified as the second and third action arguments, respectively), and *s* or *S* indicate the font selection (as made by programs such as *xfontsel(1)*) indicated by the second action argument.

start-extend()

This action is similar to **select-start** except that the selection is extended to the current pointer location.

start-cursor-extend()
This action is similar to **select-extend** except that the selection is extended to the current text cursor position.

string(*string*)
This action inserts the specified text string as if it had been typed. Quotation is necessary if the string contains whitespace or non-alphanumeric characters. If the string argument begins with the characters "0x", it is interpreted as a hex character constant.

scroll-back(*count* [,*units*])
This action scrolls the text window backward so that text that had previously scrolled off the top of the screen is now visible. The *count* argument indicates the number of *units* (which may be *page*, *halfpage*, *pixel*, or *line*) by which to scroll.

scroll-forw(*count* [,*units*])
This action scrolls is similar to **scroll-back** except that it scrolls the other direction.

allow-send-events(*on/off/toggle*)
This action set or toggles the **allowSendEvents** resource and is also invoked by the **allowsends** entry in *mainMenu*.

redraw()
This action redraws the window and is also invoked by the *redraw* entry in *mainMenu*.

send-signal(*signame*)
This action sends the signal named by *signame* to the *xterm* subprocess (the shell or program specified with the *–e* command line option) and is also invoked by the **suspend**, **continue**, **interrupt**, **hangup**, **terminate**, and **kill** entries in *mainMenu*. Allowable signal names are (case is not significant): *tstp* (if supported by the operating system), *suspend* (same as *tstp*), *cont* (if supported by the operating system), *int*, *hup*, *term*, *quit*, *alrm*, *alarm* (same as *alrm*) and *kill*.

quit() This action sends a SIGHUP to the subprogram and exits. It is also invoked by the **quit** entry in *mainMenu*.

set-scrollbar(*on/off/toggle*)
This action toggles the **scrollbar** resource and is also invoked by the **scrollbar** entry in *vtMenu*.

set-jumpscroll(*on/off/toggle*)
This action toggles the **jumpscroll** resource and is also invoked by the **jumpscroll** entry in *vtMenu*.

set-reverse-video(*on/off/toggle*)
This action toggles the *reverseVideo* resource and is also invoked by the **reversevideo** entry in *vtMenu*.

set-autowrap(*on/off/toggle*)
This action toggles automatic wrapping of long lines and is also invoked by the **autowrap** entry in *vtMenu*.

set-reversewrap(*on/off/toggle*)
This action toggles the **reverseWrap** resource and is also invoked by the **reversewrap** entry in *vtMenu*.

set-autolinefeed(*on/off/toggle*)
This action toggles automatic insertion of linefeeds and is also

invoked by the **autolinefeed** entry in *vtMenu*.

set-appcursor(*on/off/toggle*)
This action toggles the handling Application Cursor Key mode and is also invoked by the **appcursor** entry in *vtMenu*.

set-appkeypad(*on/off/toggle*)
This action toggles the handling of Application Keypad mode and is also invoked by the **appkeypad** entry in *vtMenu*.

set-scroll-on-key(*on/off/toggle*)
This action toggles the **scrollKey** resource and is also invoked from the **scrollkey** entry in *vtMenu*.

set-scroll-on-tty-output(*on/off/toggle*)
This action toggles the **scrollTtyOutput** resource and is also invoked from the **scrollttyoutput** entry in *vtMenu*.

set-allow132(*on/off/toggle*)
This action toggles the **c132** resource and is also invoked from the **allow132** entry in *vtMenu*.

set-cursesemul(*on/off/toggle*)
This action toggles the **curses** resource and is also invoked from the **cursesemul** entry in *vtMenu*.

set-visual-bell(*on/off/toggle*)
This action toggles the **visualBell** resource and is also invoked by the **visualbell** entry in *vtMenu*.

set-marginbell(*on/off/toggle*)
This action toggles the **marginBell** resource and is also invoked from the **marginbell** entry in *vtMenu*.

set-altscreen(*on/off/toggle*)
This action toggles between the alternate and current screens.

soft-reset()
This action resets the scrolling region and is also invoked from the **softreset** entry in *vtMenu*.

hard-reset()
This action resets the scrolling region, tabs, window size, and cursor keys and clears the screen. It is also invoked from the **hardreset** entry in *vtMenu*.

clear-saved-lines()
This action does **hard-reset()** (see above) and also clears the history of lines saved off the top of the screen. It is also invoked from the **clearsavedlines** entry in *vtMenu*.

set-terminal-type(*type*)
This action directs output to either the *vt* or *tek* windows, according to the *type* string. It is also invoked by the **tekmode** entry in *vtMenu* and the **vtmode** entry in *tekMenu*.

set-visibility(*vt/tek,on/off/toggle*)
This action controls whether or not the *vt* or *tek* windows are visible. It is also invoked from the **tekshow** and **vthide** entries in *vtMenu* and the **vtshow** and **tekhide** entries in *tekMenu*.

set-tek-text(*large/2/3/small*)
This action sets font used in the Tektronix window to the value of the resources **tektextlarge**, **tektext2**, **tektext3**, and **tektextsmall** according to the argument. It is also by the entries of the same names as the resources in *tekMenu*.

tek-page()
This action clears the Tektronix window and is also invoked by the **tekpage** entry in *tekMenu*.

tek-reset()
This action resets the Tektronix window and is also invoked by the *tekreset* entry in *tekMenu*.

tek-copy()
This action copies the escape codes used to generate the current window contents to a file in the current directory beginning with the name COPY. It is also invoked from the *tekcopy* entry in *tekMenu*.

visual-bell()
This action flashes the window quickly.

The Tektronix window also has the following action:

gin-press(*l/L/m/M/r/R***)**
This action sends the indicated graphics input code.

The default bindings in the VT102 window are:

```
    Shift <KeyPress> Prior:      scroll-back(1,halfpage) \n\
     Shift <KeyPress> Next:      scroll-forw(1,halfpage) \n\
  Shift <KeyPress> Select:       select-cursor-start() \
                                 select-cursor-end(PRIMARY, CUT_BUFFER0) \n\
  Shift <KeyPress> Insert:       insert-selection(PRIMARY, CUT_BUFFER0) \n\
          ~Meta<KeyPress>:       insert-seven-bit() \n\
           Meta<KeyPress>:       insert-eight-bit() \n\
         !Ctrl <Btn1Down>:       popup-menu(mainMenu) \n\
   !Lock Ctrl <Btn1Down>:        popup-menu(mainMenu) \n\
        ~Meta <Btn1Down>:        select-start() \n\
      ~Meta <Btn1Motion>:        select-extend() \n\
         !Ctrl <Btn2Down>:       popup-menu(vtMenu) \n\
   !Lock Ctrl <Btn2Down>:        popup-menu(vtMenu) \n\
  ~Ctrl ~Meta <Btn2Down>:        ignore() \n\
    ~Ctrl ~Meta <Btn2Up>:        insert-selection(PRIMARY, CUT_BUFFER0) \n\
         !Ctrl <Btn3Down>:       popup-menu(fontMenu) \n\
   !Lock Ctrl <Btn3Down>:        popup-menu(fontMenu) \n\
  ~Ctrl ~Meta <Btn3Down>:        start-extend() \n\
      ~Meta <Btn3Motion>:        select-extend() \n\
                                         <BtnUp>:select-end(PRIMARY, CUT_BUF
                 <BtnDown>:      bell(0)
```

The default bindings in the Tektronix window are:

```
        ~Meta<KeyPress>:        insert-seven-bit() \n\
         Meta<KeyPress>:        insert-eight-bit() \n\
       !Ctrl <Btn1Down>:        popup-menu(mainMenu) \n\
  !Lock Ctrl <Btn1Down>:        popup-menu(mainMenu) \n\
       !Ctrl <Btn2Down>:        popup-menu(tekMenu) \n\
  !Lock Ctrl <Btn2Down>:        popup-menu(tekMenu) \n\
  Shift ~Meta<Btn1Down>:        gin-press(L) \n\
        ~Meta<Btn1Down>:        gin-press(l) \n\
  Shift ~Meta<Btn2Down>:        gin-press(M) \n\
        ~Meta<Btn2Down>:        gin-press(m) \n\
  Shift ~Meta<Btn3Down>:        gin-press(R) \n\
        ~Meta<Btn3Down>:        gin-press(r)
```

Below is a sample how of the **keymap()** action is used to add special keys for entering commonly-typed works:

```
*VT100.Translations: #override <Key>F13: keymap(dbx)
*VT100.dbxKeymap.translations: \
      <Key>F14:   keymap(None) \n\
      <Key>F17:   string("next") string(0x0d) \n\
      <Key>F18:   string("step") string(0x0d) \n\
      <Key>F19:   string("continue") string(0x0d) \n\
      <Key>F20:   string("print ") insert-selection(PRIMARY, CUT_BUFFER0)
```

ENVIRONMENT

Xterm sets the environment variables *TERM* and *TERMCAP* properly for the size window you have created. It also uses and sets the environment variable *DISPLAY* to specify which bit map display terminal to use. The environment variable *WINDOWID* is set to the X window id number of the *xterm* window.

SEE ALSO

resize, *X*

pty, *tty*

Xterm Control Sequences (in the *xterm* source directory)

BUGS

Large pastes do not work on some systems. This is not a bug in *xterm*; it is a bug in the pseudo terminal driver of those systems. *xterm* feeds large pastes to the pty only as fast as the pty will accept data, but some pty drivers do not return enough information to know if the write has succeeded.

Many of the options are not resettable after *xterm* starts.

The Tek widget does not support key / button re-binding.

Only fixed-width, character-cell fonts are supported.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

There needs to be a dialog box to allow entry of the Tek COPY file name.

COPYRIGHT

Copyright 1989, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHORS

Far too many people, including:

Loretta Guarino Reid (DEC-UEG-WSL), Joel McCormack (DEC-UEG-WSL), Terry Weissman (DEC-UEG-WSL), Edward Moy (Berkeley), Ralph R. Swick (MIT-Athena), Mark Vandevoorde (MIT-Athena), Bob McNamara (DEC-MAD), Jim Gettys (MIT-Athena), Bob Scheifler (MIT X Consortium), Doug Mink (SAO), Steve Pitschke (Stellar), Ron Newman (MIT-Athena), Jim Fulton (MIT X Consortium), Dave Serisky (HP), Jonathan Kamens (MIT-Athena)

NAME

xwd – dump an image of an X window

SYNOPSIS

```
xwd [-add value] [-debug] [-display display]
    [-frame] [-help] [-icmap] [-nobdrs]
    [-out file] [-root | -id id | -name name]
    [-screen] [-xy]
```

DESCRIPTION

Xwd is an X Window System window dumping utility. *Xwd* allows X users to store window images in a specially formatted dump file. This file can then be read by various other X utilities for redisplay, printing, editing, formatting, archiving, image processing, etc. The target window is selected by clicking the pointer in the desired window. The keyboard bell is rung once at the beginning of the dump and twice when the dump is completed.

OPTIONS

–add *value*
This option specifies an signed value to be added to every pixel.

–frame
This option indicates that the window manager frame should be included when manually selecting a window.

–display *display*
This argument allows you to specify the server to connect to; see *X(1)*.

–id *id*
This option indicates that the window with the specified resource id should be selected for the window dump, without requiring the user to select a window with the pointer.

–icmap
Normally the colormap of the chosen window is used to obtain RGB values. This option forces the first installed colormap of the screen to be used instead.

–help Print out the 'Usage:' command syntax summary.

–name *name*
This option indicates that the window with the specified WM_NAME property should be selected for the window dump, without requiring the user to select a window with the pointer.

–nobdrs
This argument specifies that the window dump should not include the pixels that compose the X window border. This is useful in situations where you may wish to include the window contents in a document as an illustration.

–out *file*
This argument allows the user to explicitly specify the output file on the command line. The default is to output to standard out.

–root
This option indicates that the root window should be selected for the window dump, without requiring the user to select a window with the pointer.

–screen
This option indicates that the GetImage request used to obtain the image should be done on the root window, rather than directly on the specified window. In this way, you can obtain pieces of other windows that overlap the specified window, and more importantly, you can capture menus or other popups that are independent windows but appear over the specified window.

–xy This option applies to color displays only. It selects 'XY' format dumping instead of the default 'Z' format.

ENVIRONMENT VARIABLES

DISPLAY
Contains default host and display number.

FILES

XWDFile.h
X Window Dump File format definition file.

SEE ALSO
xwud, *lp*, *X*

COPYRIGHT
Copyright 1988, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHORS
Tony Della Fera, Digital Equipment Corp., MIT Project Athena
William F. Wyatt, Smithsonian Astrophysical Observatory

NAME

xwininfo – window information utility for X

SYNOPSIS

```
xwininfo [-all] [-bits] [-children] [-display display]
         [-english] [-events] [-frame] [-help] [-id id]
         [-int] [-metric] [-name name] [-root] [-shape]
         [-size] [-stats] [-tree] [-wm]
```

DESCRIPTION

Xwininfo is a utility for displaying information about windows. Various information is displayed depending on which options are selected. If no options are chosen, **–stats** is assumed.

The user has the option of selecting the target window with the mouse (by clicking any mouse button in the desired window) or by specifying its window id on the command line with the **–id** option. Or instead of specifying the window by its id number, the **–name** option may be used to specify which window is desired by name. There is also a special **–root** option to quickly obtain information on the screen's root window.

OPTIONS

–all

Display all information possible.

–bits

Display various attributes pertaining to the selected window's raw bits and how the selected window is to be stored. Displayed information includes the selected window's bit gravity, window gravity, backing-store hint, backing-planes value, backing pixel, and whether or not the window has save-under set.

–children

Display the root, parent, and children windows' ids and names of the selected window.

–display *display*

Connect to specified server. See *X*.

–english

Display all individual height, width, and x and y positions in inches (and feet, yards, and miles if necessary) as well as number of pixels. **–metric** and **–english** may both be enabled at the same time.

–events

Display the selected window's event masks. Both the event mask of events wanted by some client and the event mask of events not to propagate are displayed.

–frame

Consider window manager frames when manually selecting windows.

–help Print out the 'Usage:' command syntax summary.

–id *id* Specify target window *id* on the command line rather than using the moue to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the mouse might be impossible or interfere with the application.

–int
Display all X window IDs as integer decimal values. The default is to display them as hexadecimal values.

–metric
Display all individual height, width, and x and y positions in millimeters as well as number of pixels, based on what the server thinks the resolution is. Geometry specifications that are in **+x+y** form are not changed.

–name *name*
Specify the window named *name* as the target window on the command line rather than using the mouse to select the target window.

–root
Use the root window as the target window. This is useful in situations where the root window is completely obscured.

–shape
Display the window and border shape extents of the selected window.

–size
Display the selected window's sizing hints. Displayed information includes: for both the normal size hints and the zoom size hints, the user supplied location if any; the program supplied location if any; the user supplied size if any; the program supplied size if any; the minimum size if any; the maximum size if any; the resize increments if any; and the minimum and maximum aspect ratios if any.

–stats
Display various attributes pertaining to the location and appearance of the selected window. information displayed includes the location of the window, its width and height, its depth, border width, class, colormap id if any, map state, backing-store hint, and location of the corners.

–tree
Like **–children** but displays all children recursively.

–wm
Display the selected window's window manager hints. Information displayed may include whether or not the application accepts input, what the window's icon window # and name is, where the window's icon should go, and what the window's initial state should be.

EXAMPLE

The following is a sample summary taken with no options specified:

```
xwininfo: Window id: 0x60000f "xterm"

  Absolute upper-left X: 2
  Absolute upper-left Y: 85
  Relative upper-left X:  0
  Relative upper-left Y:  25
  Width: 579
  Height: 316
  Depth: 8
  Visual Class: PseudoColor
  Border width: 0
  Class: InputOutput
  Colormap: 0x27 (installed)
  Bit Gravity State: NorthWestGravity
  Window Gravity State: NorthWestGravity
  Backing Store State: NotUseful
  Save Under State: no
  Map State: IsViewable
  Override Redirect State: no
  Corners:  +2+85  -699+85  -699-623  +2-623
  -geometry 80x24+0+58
```

ENVIRONMENT VARIABLES

DISPLAY
Contains the default host and display number.

SEE ALSO

X, *xprop*

BUGS

Using **–stats –bits** shows some redundant information.

The **–geometry** string displayed must make assumptions about the window's border width and the behavior of the application and the window manager. As a result, the location given is not always correct.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xwud – image displayer for X

SYNOPSIS

```
xwud [-in file] [-noclick] [-geometry geom] [-display display]
     [-new] [-std maptype] [-raw] [-vis vis-type-or-id]
     [-help] [-rv] [-plane number] [-fg color] [-bg color]
```

DESCRIPTION

Xwud is an X Window System image undumping utility. *Xwud* allows X users to display in a window an image saved in a specially formatted dump file, such as produced by *xwd*.

OPTIONS

–bg *color*
If a bitmap image (or a single plane of an image) is displayed, this option can be used to specify the color to display for the "0" bits in the image.

–display *display*
This option allows you to specify the server to connect to; see *X(1)*.

–fg *color*
If a bitmap image (or a single plane of an image) is displayed, this option can be used to specify the color to display for the "1" bits in the image.

–geometry *geom*
This option allows you to specify the size and position of the window. Typically you will only want to specify the position, and let the size default to the actual size of the image.

–help Print out a short description of the allowable options.

–in *file*
This option allows the user to explicitly specify the input file on the command line. If no input file is given, the standard input is assumed.

–new This option forces creation of a new colormap for displaying the image. If the image characteristics happen to match those of the display, this can get the image on the screen faster, but at the cost of using a new colormap (which on most displays will cause other windows to go technicolor).

–noclick
Clicking any button in the window will terminate the application, unless this option is specified. Termination can always be achieved by typing 'q', 'Q', or ctrl-c.

–plane *number*
You can select a single bit plane of the image to display with this option. Planes are numbered with zero being the least significant bit. This option can be used to figure out which plane to pass to *xpr* for printing.

–raw
This option forces the image to be displayed with whatever color values happen to currently exist on the screen. This option is mostly useful when undumping an image back onto the same screen that the image originally came from, while the original windows are still on the screen, and results in getting the image on the screen faster.

–rv If a bitmap image (or a single plane of an image) is displayed, this option forces the foreground and background colors to be swapped. This may be needed when displaying a bitmap image which has the color sense of pixel values "0" and "1" reversed from what they are on your display.

–std *maptype*
This option causes the image to be displayed using the specified Standard Colormap. The property name is obtained by converting the type to upper case, prepending "RGB_", and appending "_MAP". Typical types are "best", "default", and "gray". See *xstdcmap(1)* for one way of creating Standard Colormaps.

–vis *vis-type-or-id*
This option allows you to specify a particular visual or visual class. The default is to pick the "best" one. A particular class can be specified: "StaticGray", "GrayScale", "StaticColor", "PseudoColor", "DirectColor", or "TrueColor". Or "Match" can be specified, meaning use the same class as the source image. Alternatively, an exact visual id (specific to the server) can be specified, either as a hexadecimal number (prefixed with "0x") or as a decimal number. Finally, "default" can be specified, meaning to use the same class as the colormap of the root window. Case is not significant in any of these strings.

ENVIRONMENT VARIABLES

DISPLAY
Contains default display.

FILES

XWDFile.h
X Window Dump File format definition file.

SEE ALSO

xwd, *xpr*, *xstdcmap*, *X*

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X* for a full statement of rights and permissions.

AUTHOR

Bob Scheifler, MIT X Consortium

NAME

yacc – an LALR(1) parser generator

SYNOPSIS

yacc **[**-dlrtv**]** **[**-b *file_prefix***]** **[**-p *symbol_prefix***]** *filename*

DESCRIPTION

Yacc reads the grammar specification in the file *filename* and generates an LR(1) parser for it. The parsers consist of a set of LALR(1) parsing tables and a driver routine written in the C programming language. *Yacc* normally writes the parse tables and the driver routine to the file **y.tab.c**.

The following options are available:

–b *file_prefix*
: Use *file_prefix* as the prefix prepended to the output file names instead of the character y.

–d
: Write the header file option causes the header file **y.tab.h**.

–l
: Do not insert **#line** directives in the generated code. The **#line** directives let the C compiler relate errors in the generated code to the user's original code. Any **#line** directives specified by the user will be retained. By default, the directives are inserted.

–p *symbol_prefix*
: use *symbol_prefix* as the prefix prepended to *yacc*-generated symbols. The default prefix is the string "yy".

–r
: Produce separate files for code and tables. The code file is named **y.code.c**, and the tables file is named **y.tab.c**.

–t
: Change the preprocessor directives generated by *yacc* so that debugging statements will be incorporated in the compiled code.

–v
: Write a human-readable description of the generated parser to the file **y.output**.

FORMAT OF THE INPUT FILE

The *yacc* input file consists of three sections, separated by a line with just **%%** in it:

definitions
%%
rules
%%
user code

The definitions section contains declarations; it can also contain C comments (delimited by **/*** and ***/**), or a literal block of C code, copied to the beginning of the generated file. This literal block usually contains declaration and #include lines. The following keywords can be used in the definitions section:

%{...%}
: Delimits a literal block of C code. The literal block usually contains declarations of variables and functions used by the code in the rules section.

%left *operator*
Declares an operator left-associative. Operators must be declared in increasing order of precedence.

%nonassoc *operator*
Declares an operator non-associative. Operators must be declared in increasing order of precedence.

%right *operator*
Declares an operator right-associative. Operators must be declared in increasing order of precedence.

%start *rulename*
Declares the first rule the parser should start parsing. Normally, this is the first rule in the rules section, but this declaration explicitly labels a different rule.

%token *name ...*
Defines a symbolic token, a terminal symbol (one that the parser will not attempt to reduce). They are represented internally by integer values; you can assign a value directly to a token with the **%token** directive, but this is not recommended. Other tokens are individual characters in single quotes or are defined by **%left**, **%right**, and **%nonassoc**.

%type *type name,name,...*
Declares a non-terminal token as a particular *type*. The *type* must already have been defined by a **%union**.

%union
Identify all possible C types a symbol value can have; this union is declared as type YYSTYPE in the generated source file. The format is:

```
%union {
        ...field declarations
}
```

The rules section is the heart of the grammar. Each rule starts with a non-terminal symbol and a colon, followed by a list of symbols, literal tokens, and actions. The list can be empty. For example, this rule states that a time is an hour, a minute, and a second value, joined by colons:

```
time: hour ':' minute ':' second ;
```

(This assumes we've already defined hour, minute, and second.) The semicolon at the end is traditional, but optional. If consecutive rules have the same left-hand side, rules after the first can start with a vertical bar, rather than the name and a colon. This is somewhat easier to read — but the semicolon must be omitted before the vertical bar.

An *action* in a rule is a C compound statement to be executed whenever the parser reaches that point in the grammar. In the action, the name **$$** stands for the symbol on the left-hand side, and a dollar sign followed by a digit *n* stands for the *n*th simbol on the right-hand side. In the example above:

```
time: hour ':' minute ':' second
       {printf("time is %d:%d:%d\n",$1, $3, $5); }
       ;
```

If no action is defined, the action is:

```
{ $$ = $1; }
```

The user subroutines section contains routines called from the actions. It is copied directly into the C file.

ENVIRONMENT VARIABLES

Yacc makes use of the following environment variables, if set:

TMPDIR
: If set, the contents of this variable will be used as the name of the directory where the temporary files are created.

FILES

The *yacc* utility makes use of the following files:

y.code.c
: Code file created by *yacc* when the **–r** option is given.

y.tab.c Table file created by *yacc*.

y.tab.h Table header file created by *yacc*.

y.output
: A human-readable version of the output, created with the **–v** option.

$OPENNT_ROOT/tmp/yacc.a*XXXXXX*
$OPENNT_ROOT/tmp/yacc.t*XXXXXX*
$OPENNT_ROOT/tmp/yacc.u*XXXXXX*
: Temporary files.

DIAGNOSTICS

If there are rules that are never reduced, the number of such rules is reported on standard error. If there are any LALR(1) conflicts, the number of conflicts is reported on standard error.

NAME

zip – package and compress (archive) files

SYNOPSIS

```
zip [-cdDeEfFghjklLmoqrSTuvVwyz@] [-b path] [-n suffixes]
    [-t mmddyy] [zipfile [file1 file2 ...]] [-xi list]
```

DESCRIPTION

zip is a compression and file packaging utility for many different operating systems including INTERIX. It is analogous to a combination of the INTERIX utilities *tar* and *compress* and is compatible with PKZIP (Phil Katz's ZIP for MS-DOS systems).

The companion program *unzip* unpacks *zip* archives. The *zip* and *unzip* programs can work with archives produced by PKZIP, and PKZIP and PKUNZIP can work with archives produced by *zip*. *zip* version 2.0.1 is compatible with PKZIP 2.04 Note that PKUNZIP 1.10 cannot extract files produced by PKZIP 2.04 or *zip* 2.0.1. You must use PKUNZIP 2.04g or *unzip* 5.0p1 (or later versions) to extract them.

For a brief help on *zip* and *unzip*, run each without specifying any parameters on the command line.

The program is useful for packaging a set of files for distribution; for archiving files; and for saving disk space by temporarily compressing unused files or directories.

Zip accepts the following options:

–b *path* Use the specified *path* for the temporary *zip* archive. For example:

```
zip -b /tmp stuff *
```

will put the temporary *zip* archive in the directory **/tmp**, copying over **stuff.zip** to the current directory when done. This option is only useful when updating an existing archive and the file system containing this old archive does not have enough space to hold both old and new archive at the same time.

–c Add one-line comments for each file. File operations (adding, updating) are done first, and the user is then prompted for a one-line comment for each file. Enter the comment followed by return, or just return for no comment.

–d Remove (delete) entries from a *zip* archive. For example:

```
zip -d foo foo/tom/junk foo/harry/\* \*.o
```

will remove the entry **foo/tom/junk**, all of the files that start with **foo/harry/**, and all of the files that end with **.o** (in any path). Note that shell pathname expansion has been inhibited with backslashes, so that *zip* can see the asterisks, enabling *zip* to match on the contents of the *zip* archive instead of the contents of the current directory.

–D Do not create entries in the *zip* archive for directories. Directory entries are created by default so that their attributes can be saved in the zip archive. The environment variable ZIPOPT can be used to change the default options. For example with sh (the Bourne shell):

```
ZIPOPT="-D"; export ZIPOPT
```

(The variable *ZIPOPT* can be used for any option except **–i** and **–x** and can include several options.) The option **–D** is a shorthand for **–x** "*/" but the latter cannot be set as default in the ZIPOPT environment variable.

–e Encrypt the contents of the *zip* archive using a password which is entered on the terminal in response to a prompt (this will not be echoed; if standard error is not a tty, *zip* will exit with an error).

–ee Encrypt contents, prompting for the password *twice*, checking that the two entries are identical before using the password.

–f Replace (freshen) an existing entry in the *zip* archive only if it has been modified more recently than the version already in the *zip* archive; unlike the update option (**–u**) this will not add files that are not already in the *zip* archive. For example:

```
zip -f foo
```

This command should be run from the same directory from which the original *zip* command was run, since paths stored in *zip* archives are always relative.

–F Fix the *zip* archive. This option can be used if some portions of the archive are missing. It is not guaranteed to work, so you MUST make a backup of the original archive first.

When doubled as in **–FF** the compressed sizes given inside the damaged archive are not trusted and zip scans for special signatures to identify the limits between the archive members. The single **–F** is more reliable if the archive is not too much damaged, for example if it has only been truncated, so try this option first.

Neither option will recover archives that have been incorrectly transferred in ascii mode instead of binary. After the repair, the **–t** option of *unzip* may show that some files have a bad CRC. Such files cannot be recovered; you can remove them from the archive using the **–d** option of *zip*.

–g Grow (append to) the specified *zip* archive, instead of creating a new one. If this operation fails, *zip* attempts to restore the archive to its original state. If the restoration fails, the archive might become corrupted.

–h Display the *zip* help information (this also appears if *zip* is run with no arguments).

–i *files*

Include only the specified *files*, as in:

```
zip -r foo . -i \*.c
```

which will include only the files that end in **.c** in the current directory and its subdirectories. (Note for PKZIP users: the equivalent command is

```
pkzip -r foo *.c
```

PKZIP does not allow recursion in directories other than the current one.) The backslash avoids the shell filename substitution, so that the name matching is performed by *zip* at all directory levels.

–j Store just the name of a saved file (junk the path), and do not store directory names. By default, *zip* will store the full path (relative to the current path).

–k Attempt to convert the names and paths to conform to MS-DOS, store only the MS-DOS attribute (just the user write attribute from INTERIX), and mark the entry as made under MS-DOS (even though it was not); for compatibility with PKUNZIP under MS-DOS which cannot handle certain names such as those with two dots.

–l Translate the INTERIX end-of-line character LF into the MS-DOS convention CR-LF. This option should not be used on binary files. This option can be used on open and POSIX systems if the zip file is intended for PKUNZIP under MS-DOS. If the input files already contain CR LF, this option adds an extra CR. This ensures that **unzip –a** on open and POSIX systems will get back an exact copy of the original file, to undo the effect of **zip –l**.

–ll Translate the MS-DOS end-of-line CR LF into POSIX (open systems) LF. This option should not be used on binary files. This option can be used on MS-DOS if the zip file is intended for *unzip* under open and POSIX systems.

–L Display the *zip* license.

–m Move the specified files into the *zip* archive; actually, this deletes the target directories / files after making the specified *zip* archive. If a directory becomes empty after removal of the files, the directory is also removed. No deletions are done until *zip* has created the archive without error. This is useful for conserving disk space, but is potentially dangerous so it is recommended to use it in combination with **–T** to test the archive before removing all input files.

–n *suffixes* Do not attempt to compress files named with the given *suffixes*. Such files are simply stored (0% compression) in the output zip file, so that *zip* doesn't waste its time trying to compress them. The suffixes are separated by either colons or semicolons. For example:

```
zip -rn .Z:.zip:.tiff:.gif:.snd  foo foo
```

will copy everything from *foo* into **foo.zip**, but will store any files that end in **.Z**, **.zip**, **.tiff**, **.gif**, or **.snd** without trying to compress them (image and sound files often have their own specialized compression methods). By default, *zip* does not compress files with extensions in the list *.Z*:*.zip*:*.zoo*:*.arc*:*.lzh*:*.arj*. Such files are stored directly in the

output archive. The environment variable *ZIPOPT* can be used to change the default options. For example under INTERIX with csh:

```
setenv ZIPOPT "-n .gif:.zip"
```

To attempt compression on all files, use:

```
zip -n : foo
```

The maximum compression option **–9** also attempts compression on all files regardless of extension.

–o Set the "last modified" time of the *zip* archive to the latest (oldest) "last modified" time found among the entries in the *zip* archive. This can be used without any other operations, if desired. For example:

```
zip -o foo
```

will change the last modified time of **foo.zip** to the latest time of the entries in **foo.zip**.

–q Quiet mode; eliminate informational messages and comment prompts. (Useful, for example, in shell scripts and background tasks).

–r Travel the directory structure recursively; for example:

```
zip -r foo foo
```

In this case, all the files and directories in **foo** are saved in a *zip* archive named **foo.zip**, including files with names starting with ".", since the recursion does not use the shell's file-name substitution mechanism. If you wish to include only a specific subset of the files in directory **foo** and its subdirectories, use the **–i** option to the specify the pattern of files to be included. You should not use **–r** with the name ".*", since that matches ".." which will attempt to zip up the parent directory (probably not what was intended).

–S Include system and hidden files. This option is effective on some systems only; it is ignored on INTERIX.

–t *mmddyy* Do not operate on files modified prior to the specified date, where *mm* is the month (0-12), *dd* is the day of the month (1-31), and *yy* are the last two digits of the year. For example:

```
zip -rt 120791 infamy foo
```

will add all the files in **foo** and its subdirectories that were last modified on or after 7 December 1991, to the *zip* archive *infamy.zip*.

–T Test the integrity of the new zip file. If the check fails, the old zip file is unchanged and (with the **–m** option) not input files are removed.

–u Replace (update) an existing entry in the *zip* archive only if it has been modified more recently than the version already in the *zip* archive. For example:

```
zip -u stuff *
```

will add any new files in the current directory, and update any files which have been modified since the *zip* archive **stuff.zip** was last created / modified (note that *zip* will not try to pack *stuff.zip* into itself

when you do this).

Note that the **–u** option with no arguments acts like the **–f** (freshen) option.

–v Verbose mode. Display a progress indicator during compression.

–V Save VMS file attributes. This option is available on VMS only; *zip* archives created with this option will generally not be usable on other systems.

–w Append the version number of the files to the name, including multiple versions of files. (VMS only; default: use only the most recent version of a specified file).

–x *files* Explicitly exclude the specified *files*, as in:

```
zip -r foo foo -x \*.o
```

which will include the contents of **foo** in **foo.zip** while excluding all the files that end in **.o**. The backslash avoids the shell filename substitution, so that the name matching is performed by *zip* at all directory levels.

–y Store symbolic links as such in the *zip* archive, instead of compressing and storing the file referred to by the link.

–z Prompt for a multi-line comment for the entire *zip* archive. The comment is ended by a line containing just a period, or an end of file condition (^D on INTERIX, ^Z on MS-DOS, OS/2, and VAX/VMS). The comment can be taken from a file:

```
zip -z foo < foowhat
```

–# Regulate the speed of compression using the specified digit #, where **–0** indicates no compression (store all files), **–1** indicates the fastest compression method (less compression) and **–9** indicates the slowest compression method (optimal compression, ignores the suffix list). The default compression level is **–6**.

–@ Take the list of input files from standard input.

The *zip* program puts one or more compressed files into a single *zip* archive, along with information about the files (name, path, date, time of last modification, protection, and check information to verify file integrity). An entire directory structure can be packed into a *zip* archive with a single command. Compression ratios of 2:1 to 3:1 are common for text files. *zip* has one compression method (deflation) and can also store files without compression. *zip* automatically chooses the better of the two for each file to be compressed.

When given the name of an existing *zip* archive, *zip* will replace identically named entries in the *zip* archive or add entries for new names. For example, if **foo.zip** exists and contains **foo/file1** and **foo/file2**, and the directory **foo** contains the files **foo/file1** and **foo/file3**, then:

```
zip -r foo foo
```

will replace **foo/file1** in **foo.zip** and add **foo/file3** to **foo.zip**. After this, **foo.zip** contains **foo/file1**, **foo/file2**, and **foo/file3**, with **foo/file2** unchanged from before.

If the file list is specified as **–@**, *zip* takes the list of input files from standard input. Under INTERIX, this option can be used to powerful effect in conjunction with the *find* command. For example, to archive all the C source files in the current directory and its subdirectories:

```
find . -name "*.[ch]" -print | zip source -@
```

(note that the pattern must be quoted to keep the shell from expanding it). *zip* will also accept a single dash ("-") as the zip file name, in which case it will write the zip file to standard output, allowing the output to be piped to another program. For example:

```
zip -r - . | dd of=/dev/nrst0 obs=16k
```

would write the zip output directly to a tape with the specified block size for the purpose of backing up the current directory.

zip also accepts a single dash ("-") as the name of a file to be compressed, in which case it will read the file from standard input, allowing zip to take input from another program. For example:

```
tar cf - . | zip backup -
```

would compress the output of the tar command for the purpose of backing up the current directory. This generally produces better compression than the previous example using the -r option, because *zip* can take advantage of redundancy between files. The backup can be restored using the command

```
unzip -p backup | tar xf -
```

When no zip file name is given and stdout is not a terminal, *zip* acts as a filter, compressing standard input to standard output. For example,

```
tar cf - . | zip | dd of=/dev/nrst0 obs=16k
```

is equivalent to

```
tar cf - . | zip - - | dd of=/dev/nrst0 obs=16k
```

zip archives created in this manner can be extracted with the program *gunzip* which is provided in the *gzip* package. For example:

```
dd if=/dev/nrst0  ibs=16k | gunzip | tar xvf -
```

When changing an existing *zip* archive, *zip* will write a temporary file with the new contents, and only replace the old one when the process of creating the new version has been completed without error.

If the name of the *zip* archive does not contain an extension, the extension **.zip** is added. If the name already contains an extension other than **.zip** the existing extension is kept unchanged.

EXAMPLES

The simplest example:

```
zip stuff *
```

creates the archive **stuff.zip** (assuming it does not exist) and puts all the files in the current directory in it, in compressed form (the **.zip** suffix is added automatically, unless that archive name given contains a dot already; this allows the explicit specification of other suffixes).

Because of the way the shell does filename substitution, files starting with "." are not included; to include these as well:

```
zip stuff .* *
```

Even this will not include any subdirectories from the current directory.

To zip up an entire directory, the command:

```
zip -r foo foo
```

creates the archive **foo.zip**, containing all the files and directories in the directory **foo** that is contained within the current directory.

You may want to make a *zip* archive that contains the files in **foo**, without recording the directory name, **foo**. You can use the **–j** option to leave off the paths, as in:

```
zip -j foo foo/*
```

If you are short on disk space, you might not have enough room to hold both the original directory and the corresponding compressed *zip* archive. In this case, you can create the archive in steps using the **–m** option. If *foo* contains the subdirectories *tom*, *dick*, and *harry*, you can:

```
zip -rm foo foo/tom
zip -rm foo foo/dick
zip -rm foo foo/harry
```

where the first command creates **foo.zip**, and the next two add to it. At the completion of each *zip* command, the last created archive is deleted, making room for the next *zip* command to function.

PATTERN MATCHING

This section applies only to open and POSIX systems. Watch this space for details on MS-DOS and VMS operation.

The standard shells (*sh* and *ksh*) do filename substitution on command arguments. The special characters are:

? match any single character

***** match any number of characters (including none)

[] match any character in the range indicated within the brackets (example: [a–f], [0–9]).

When these characters are encountered (without being escaped with a backslash or quotes), the shell will look for files relative to the current path that match the pattern, and replace the argument with a list of the names that matched.

The *zip* program can do the same matching on names that are in the *zip* archive being modified or, in the case of the **–x** (exclude) or **–i** (include) options, on the list of files to be operated on, by using backslashes or quotes to tell the shell not

to do the name expansion. In general, when *zip* encounters a name in the list of files to do, it first looks for the name in the file system. If it finds it, it then adds it to the list of files to do. If it does not find it, it looks for the name in the *zip* archive being modified (if it exists), using the pattern matching characters described above, if present. For each match, it will add that name to the list of files to be processed, unless this name matches one given with the **–x** option, or does not match any name given with the **–i** option.

The pattern matching includes the path, and so patterns like *.o match names that end in ".o", no matter what the path prefix is. Note that the backslash must precede every special character (i.e. ?*[]), or the entire argument must be enclosed in double quotes ("").

In general, use backslash to make *zip* do the pattern matching with the **–f** (freshen) and **–d** (delete) options, and sometimes after the **–x** (exclude) option when used with an appropriate operation (add, **–u**, **–f**, or **–d**).

BUGS

zip 2.0.1 is not compatible with PKUNZIP 1.10. Use *zip* 1.1 to produce *zip* files which can be extracted by PKUNZIP 1.10.

zip files produced by *zip* 2.0.1 must not be *updated* by *zip* 1.1 or PKZIP 1.10, if they contain encrypted members or if they have been produced in a pipe or on a non-seekable device. The old versions of *zip* or PKZIP would create an archive with an incorrect format. The old versions can list the contents of the zip file but cannot extract it anyway (because of the new compression algorithm). If you do not use encryption and use regular disk files, you do not have to care about this problem.

Under VMS, not all of the odd file formats are treated properly. Only stream-LF format *zip* files are expected to work with *zip*. Others can be converted using Rahul Dhesi's BILF program. This version of *zip* handles some of the conversion internally. When using Kermit to transfer zip files from Vax to MS-DOS, type "set file type block" on the Vax. When transfering from MS-DOS to Vax, type "set file type fixed" on the Vax. In both cases, type "set file type binary" on MS-DOS.

Under VMS, *zip* hangs for file specification that uses DECnet syntax *foo*::*.*.

On OS/2, zip cannot match some names, such as those including an exclamation mark or a hash sign. This is a bug in OS/2 itself: the 32-bit DosFindFirst/Next don't find such names. Other programs such as GNU tar are also affected by this bug.

Under OS/2, the amount of External Attributes displayed by DIR is (for compatibility) the amount returned by the 16-bit version of *DosQueryPathInfo()*( ). Otherwise OS/2 1.3 and 2.0 would report different EA sizes when DIRing a file. However, the structure layout returned by the 32-bit DosQueryPathInfo() is a bit different, it uses extra padding bytes and link pointers (it's a linked list) to have all fields on 4-byte boundaries for portability to future RISC OS/2 versions. Therefore the value reported by *zip* (which uses this 32-bit-mode size) differs from that reported by DIR. *zip* stores the 32-bit format for portability, even the 16-bit MS-C-compiled version running on OS/2 1.3, so even this one shows the 32-bit-mode size.

AUTHORS

Copyright (C) 1990-1993 Mark Adler, Richard B. Wales, Jean-loup Gailly, Kai Uwe Rommel, Igor Mandrichenko and John Bush. Permission is granted to any individual or institution to use, copy, or redistribute this software so long as all of the original files are included, that it is not sold for profit, and that this copyright notice is retained.

LIKE ANYTHING ELSE THAT'S FREE, ZIP AND ITS ASSOCIATED UTILITIES ARE PROVIDED AS IS AND COME WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. IN NO EVENT WILL THE COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES RESULTING FROM THE USE OF THIS SOFTWARE.

Please send bug reports and comments by email to: *zip–bugs@wkuvx1.bitnet*. For bug reports, please include the version of *zip*, the make options used to compile it, the machine and operating system in use, and as much additional information as possible.

ACKNOWLEDGEMENTS

Thanks to R. P. Byrne for his *Shrink.Pas* program, which inspired this project, and from which the shrink algorithm was stolen; to Phil Katz for placing in the public domain the *zip* file format, compression format, and .ZIP filename extension, and for accepting minor changes to the file format; to Steve Burg for clarifications on the deflate format; to Haruhiko Okumura and Leonid Broukhis for providing some useful ideas for the compression algorithm; to Keith Petersen, Rich Wales, Hunter Goatley and Mark Adler for providing a mailing list and *ftp* site for the INFO-ZIP group to use; and most importantly, to the INFO-ZIP group itself (listed in the file **infozip.who**) without whose tireless testing and bug-fixing efforts a portable *zip* would not have been possible. Finally we should thank (blame) the first INFO-ZIP moderator, David Kirschbaum, for getting us into this mess in the first place. The manual page was rewritten for troff macros by R. P. C. Rodgers.

SEE ALSO

compress, *gzip*, *shar*, *tar*, *unzip*