

System Information and File Formats

This section contains miscellaneous reference pages for the INTERIX Commands and Utilities, Release 2.2. This includes file formats and regular expression syntax.

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

crontab – tables for driving cron

DESCRIPTION

A **crontab** file contains instructions to the *cron1* daemon of the general form: “run this command at this time on this date”. Each user who has been assigned cron privileges using the *cronadmin* program has their own crontab and their own *cron1* daemon.

Blank lines and leading spaces and tabs are ignored. Lines whose first non-space character is a pound-sign (#) are comments, and are ignored. Note that comments are not allowed on the same line as cron commands, since they will be taken to be part of the command. Similarly, comments are not allowed on the same line as environment variable settings.

An active line in a crontab will be either an environment setting or a cron command. An environment setting is of the form,

name = *value*

where the spaces around the equal-sign (=) are optional, and any subsequent non-leading spaces in *value* will be part of the value assigned to *name*. The *value* string may be placed in quotes (single or double, but matching) to preserve leading or trailing blanks.

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 format of a cron command is very much the V7 standard, with a number of upward-compatible extensions. Each line has five time and date fields, followed by a command. Commands are executed by *cron1* when the minute, hour, and month of year fields match the current time, *and* when at least one of the two day fields (day of month, or day of week) match the current time (see “Note” below). *cron1* examines cron entries once every minute. The time and date fields are:

field	allowed values
minute	0-59
hour	0-23
day of month	0-31

month 0-12 (or names, see below)
 day of week 0-7 (0 or 7 is Sun, or use names)

A field may be an asterisk (*), which always stands for “first–last”.

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, 8-11 for an “hours” entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: “1,2,5,9”, “0-4,8-12”.

Step values can be used in conjunction with ranges. Following a range with “/**<number>**” specifies skips of the number’s value through the range. For example, “0-23/2” can be used in the hours field to specify command execution every other hour (the alternative in the V7 standard is “0,2,4,6,8,10,12,14,16,18,20,22”). Steps are also permitted after an asterisk, so if you want to say “every two hours”, just use “*/2”.

Names can also be used for the “month” and “day of week” fields. Use the first three letters of the particular day or month (case doesn’t matter). Ranges or lists of names are not allowed.

The “sixth” field (the rest of the line) specifies the command to be run. The entire command portion of the line, up to a newline or % character, will be executed by **/bin/sh** or by the shell specified in the *SHELL* variable of the crontab. Percent-signs (%) in the command, unless escaped with backslash (\), will be changed into newline characters, and all data after the first % will be sent to the command as standard input.

Note: The day of a command’s execution can be specified by two fields — day of month, and day of week. If both fields are restricted (ie, aren’t *), the command will be run when *either* field matches the current time. For example,

```
30 4 1,15 * 5
```

would cause a command to be run at 4:30 am on the 1st and 15th of each month, plus every Friday.

EXAMPLE CRON FILE

```
# mail any output to 'paul', even if this crontab is run as 'Administrator'
MAILTO=paul
#
# run five minutes after midnight, every day
5 0 * * *      $HOME/bin/daily.job >> $HOME/tmp/out 2>&1
# run at 2:15pm on the first of every month -- output mailed to paul
15 14 1 * *    $HOME/bin/monthly
# run at 10 pm on weekdays, annoy Joe
0 22 * * 1-5   mail -s "It's 10pm" joe%Joe,%%Where are your kids?%
23 0-23/2 * * * echo "run 23 minutes after midn, 2am, 4am ..., everyday"
5 4 * * sun    echo "run at 5 after 4 every sunday"
```

SEE ALSO

cron1, *crontab*, *cronadmin*

EXTENSIONS

When specifying day of week, both day 0 and day 7 will be considered Sunday. BSD and ATT seem to disagree about this.

Lists and ranges are allowed to co-exist in the same field. "1-3,7-9" would be rejected by ATT or BSD cron — they want to see "1-3" or "7,8,9" ONLY.

Ranges can include “steps”, so "1-9/2" is the same as "1,3,5,7,9".

Names of months or days of the week can be specified by name.

Environment variables can be set in the crontab.

Command output is mailed to the crontab owner (BSD can't do this), can be mailed to a person other than the crontab owner (SysV can't do this), or the feature can be turned off and no mail will be sent at all (SysV can't do this either).

AUTHOR

Based on work by Paul Vixie <paul@vix.com>

NAME

magic – file command’s magic number file

DESCRIPTION

The *file* command identifies the type of a file using, among other tests, a test for whether the file begins with a certain *magic number*. The file **magic** specifies what magic numbers are to be tested for, what message to print if a particular magic number is found, and additional information to extract from the file.

Each line of the file specifies a test to be performed. A test compares the data starting at a particular offset in the file with a 1-byte, 2-byte, or 4-byte numeric value or a string. If the test succeeds, a message is printed. The line consists of the following fields:

offset type test message

The fields are separated by whitespace; the *message* field is the remainder of the line.

offset

A number specifying the offset, in bytes, into the file of the data which is to be tested.

type

The type of the data to be tested. The possible values are:

byte A one-byte value.

short A two-byte value (on most systems) in this machine’s native byte order.

long A four-byte value (on most systems) in this machine’s native byte order.

string A string of bytes.

date A four-byte value interpreted as a POSIX date.

beshort A two-byte value (on most systems) in big-endian byte order.

belong A four-byte value (on most systems) in big-endian byte order.

bedate A four-byte value (on most systems) in big-endian byte order, interpreted as a POSIX date.

leshort A two-byte value (on most systems) in little-endian byte order.

lelong A four-byte value (on most systems) in little-endian byte order.

ledate A four-byte value (on most systems) in little-endian byte order, interpreted as a POSIX date.

The numeric types may optionally be followed by **&** and a numeric value, to specify that the value is to be AND’ed with the numeric value before any comparisons are done.

test

The value to be compared with the value from the file. If the type is numeric, this value is specified in C form; if it is a string, it is specified as a C string with the usual escapes permitted (e.g. `\n` for new-line).

Numeric values may be preceded by a character indicating the operation to be performed. It may be `=`, to specify that the value from the file must equal the specified value, `<`, to specify that the value from the file must be less than the

specified value, **>**, to specify that the value from the file must be greater than the specified value, **&**, to specify that the value from the file must have set all of the bits that are set in the specified value, or **^**, to specify that the value from the file must have clear any of the bits that are set in the specified value.

Numeric values are specified in C form; e.g. **13** is decimal, **013** is octal, and **0x13** is hexadecimal. to specify that any value will match. If the character is omitted, it is assumed to be **=**.

For string values, the byte string from the file must match the specified byte string. The operators **=**, **<** and **>** (but not **&**) can be applied to strings. The length used for matching is that of the string argument in the magic file. This means that a line can match any string, and then presumably print that string, by doing **>\0** (because all strings are greater than the null string).

message The message to be printed if the comparison succeeds. If the string contains a *printf()*-format specification, the value from the file (with any specified masking performed) is printed using the message as the format string.

Some file formats contain additional information which is to be printed along with the file type. A line which begins with the character **>** indicates additional tests and messages to be printed. The number of **>** on the line indicates the level of the test; a line with no **>** at the beginning is considered to be at level 0. Each line at level ***n*+1** is under the control of the line at level ***n*** most closely preceding it in the magic file. If the test on a line at level ***n*** succeeds, the tests specified in all the subsequent lines at level ***n*+1** are performed, and the messages printed if the tests succeed. The next line at level ***n*** terminates this. If the first character following the last **>** is a **(** then the string after the parenthesis is interpreted as an indirect offset. That means that the number after the parenthesis is used as a offset in the file. The value at that offset is read, and is used again as an offset in the file. Indirect offsets are of the form: **((x[.bsl])[+-][y])**. The value of ***x*** is used as an offset in the file. A byte, short or long is read at that offset depending on the **[bsl]** type specifier. To that number the value of ***y*** is added and the result is used as an offset in the file. The default type if one is not specified is long.

BUGS

The formats **long**, **belong**, **lelong**, **short**, **beshort**, **leshort**, **date**, **bedate**, and **ledate** are system-dependent; perhaps they should be specified as a number of bytes (2B, 4B, etc), since the files being recognized typically come from a system on which the lengths are invariant.

There is (currently) no support for specified-endian data to be used in indirect offsets.

SEE ALSO

file

NAME

pty – pseudo terminal driver

DESCRIPTION

The *pty* driver provides support for a device-pair termed a *pseudo terminal*. A pseudo terminal is a pair of character devices, a *master* device and a *slave* device. The slave device provides to a process an interface identical to that described in *tty*. However, whereas all other devices which provide the interface described in *tty* have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

INTERIX version 2.2 is configured with 256 pseudo-terminals.

The quick way to open a master pseudo terminal is to use *open()* with a path of */dev/ptmx*. The name of the slave side can found with *ptsname()*.

The following *ioctl()* calls apply only to pseudo terminals:

TIOCSTOP

Stops output to a terminal (e.g. like typing ^S). Takes no parameter.

TIOCPKT

Restarts output (stopped by TIOCSTOP or by typing ^S). Takes no parameter.

TIOCPKT

Enable / disable *packet* mode. Packet mode is enabled by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. When applied to the master side of a pseudo terminal, each subsequent *read* from the terminal will return data written on the slave part of the pseudo terminal preceded by a zero byte (symbolically defined as **TIOCPKT_DATA**), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

TIOCPKT_FLUSHREAD

whenever the read queue for the terminal is flushed or emptied.

TIOCPKT_FLUSHWRITE

whenever the write queue for the terminal is flushed or emptied.

TIOCPKT_STOP

whenever output to the terminal is stopped a la ^S.

TIOCPKT_START

whenever output to the terminal is restarted.

TIOCPKT_DOSTOP

whenever *c_cc[VSTART]* is ^S and *c_cc[VSTOP]* is ^Q.

TIOCPKT_NOSTOP

whenever the start and stop characters are not ^S/^Q.

While this mode is in use, the presence of control status information to be read from the master side may be detected by a *select()* for exceptional conditions.

TIOCPKT_IOCTL

whenever one of the members of the **termios** structure for the terminal changes. Only the new content of the **termios** structure is returned. Other TIOCPKT control information (except for

TIOCPKT_DATA) may be inclusively-OR'd with this control bit. A *select()* on the master side of the terminal will return true for read, in addition to exception.

While in packet mode, if a *read()* request is for fewer than **sizeof(struct termios) + 1** bytes and TIOCPKT_IOCTL control information is available, then the TIOCPKT_IOCTL information will remain pending until a sufficiently-large *read()* request is performed. All other packet mode control bits will be delivered, and *select()* will continue to return true for read and exception until the TIOCPKT_IOCTL information is retrieved.

TIOCUCNTL

Enable / disable a mode that allows a small number of simple user *ioctl* commands to be passed through the pseudo-terminal, using a protocol similar to that of TIOCPKT. The TIOCUCNTL and TIOCPKT modes are mutually exclusive.

This mode is enabled from the master side of a pseudo terminal by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. Each subsequent *read* from the master side will return data written on the slave part of the pseudo terminal preceded by a zero byte, or a single byte reflecting a user control operation on the slave side. A user control command consists of a special *ioctl* operation with no data; the command is given as *UIOCCMD*(*n*), where *n* is a number in the range 1-255. The operation value *n* will be received as a single byte on the next *read* from the master side. The *ioctl UIOCCMD*(0) is a no-op that may be used to probe for the existence of this facility. As with TIOCPKT mode, command operations may be detected with a *select* for exceptional conditions.

TIOCREMOTE

A mode for the master half of a pseudo terminal, independent of TIOCPKT. **This is not enabled in the current version of INTERIX.** This mode causes input to the pseudo terminal to be flow controlled and not input edited (regardless of the terminal mode). Each write to the control terminal produces a record boundary for the process reading the terminal. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an end-of-file character. TIOCREMOTE can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

FILES

/dev/pty[p-zA-E][0-9a-f]
master pseudo terminals
/dev/tty[p-zA-E][0-9a-f]
slave pseudo terminals

SEE ALSO

ioctl()
termios, *tty*

NAME

rcsintro – introduction to RCS commands

DESCRIPTION

The Revision Control System (RCS) manages multiple revisions of files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, graphics, papers, and form letters.

The basic user interface is extremely simple. The novice only needs to learn two commands: **ci**(1) and **co**(1). **ci**, short for check in, deposits the contents of a file into an archival file called an RCS file. An RCS file contains all revisions of a particular file. **co**, short for check out, retrieves revisions from an RCS file.

Functions of RCS

- Store and retrieve multiple revisions of text. RCS saves all old revisions in a space efficient way. Changes no longer destroy the original, because the previous revisions remain accessible. Revisions can be retrieved according to ranges of revision numbers, symbolic names, dates, authors, and states.
- Maintain a complete history of changes. RCS logs all changes automatically. Besides the text of each revision, RCS stores the author, the date and time of check-in, and a log message summarizing the change. The logging makes it easy to find out what happened to a module, without having to compare source listings or having to track down colleagues.
- Resolve access conflicts. When two or more programmers wish to modify the same revision, RCS alerts the programmers and prevents one modification from corrupting the other.
- Maintain a tree of revisions. RCS can maintain separate lines of development for each module. It stores a tree structure that represents the ancestral relationships among revisions.
- Merge revisions and resolve conflicts. Two separate lines of development of a module can be coalesced by merging. If the revisions to be merged affect the same sections of code, RCS alerts the user about the overlapping changes.
- Control releases and configurations. Revisions can be assigned symbolic names and marked as released, stable, experimental, etc. With these facilities, configurations of modules can be described simply and directly.
- Automatically identify each revision with name, revision number, creation time, author, etc. The identification is like a stamp that can be embedded at an appropriate place in the text of a revision. The identification makes it simple to determine which revisions of which modules make up a given configuration.
- Minimize secondary storage. RCS needs little extra space for the revisions (only the differences). If intermediate revisions are deleted, the corresponding deltas are compressed accordingly.

Getting Started with RCS

Suppose you have a file **f.c** that you wish to put under control of RCS. If you have not already done so, make an RCS directory with the command

```
mkdir RCS
```

Then invoke the check-in command

```
ci f.c
```

This command creates an RCS file in the **RCS** directory, stores **f.c** into it as revision 1.1, and deletes **f.c**. It also asks you for a description. The description should be a synopsis of the contents of the file. All later check-in commands will ask you for a log entry, which should summarize the changes that you made.

Files in the RCS directory are called RCS files; the others are called working files. To get back the working file **f.c** in the previous example, use the check-out command

```
co f.c
```

This command extracts the latest revision from the RCS file and writes it into **f.c**. If you want to edit **f.c**, you must lock it as you check it out with the command

```
co -l f.c
```

You can now edit **f.c**.

Suppose after some editing you want to know what changes that you have made. The command

```
rcsdiff f.c
```

tells you the difference between the most recently checked-in version and the working file. You can check the file back in by invoking

```
ci f.c
```

This increments the revision number properly.

If *ci* complains with the message

```
ci error: no lock set by your name
```

then you have tried to check in a file even though you did not lock it when you checked it out. Of course, it is too late now to do the check-out with locking, because another check-out would overwrite your modifications. Instead, invoke

```
rcs -l f.c
```

This command will lock the latest revision for you, unless somebody else got ahead of you already. In this case, you'll have to negotiate with that person.

Locking assures that you, and only you, can check in the next update, and avoids nasty problems if several people work on the same file. Even if a revision is locked, it can still be checked out for reading, compiling, etc. All that locking prevents is a *check-in* by anybody but the locker.

If your RCS file is private, i.e., if you are the only person who is going to deposit revisions into it, strict locking is not needed and you can turn it off. If strict locking is turned off, the owner of the RCS file need not have a lock for check-in; all others still do. Turning strict locking off and on is done with the commands

```
rcs -U f.c
```

```
rcs -L f.c
```

If you don't want to clutter your working directory with RCS files, create a subdirectory called **RCS** in your working directory, and move all your RCS files there. RCS commands will look first into that directory to find needed files. All the commands discussed above will still work, without any modification. (Actually, pairs of RCS and working files can be specified in three ways: (a) both are given, (b) only the working file is given, (c) only the RCS file is given. Both RCS and working files may have arbitrary path prefixes; RCS commands pair them up intelligently.)

To avoid the deletion of the working file during check-in (in case you want to continue editing or compiling), invoke

```
ci -l f.c
```

or

```
ci -u f.c
```

These commands check in **f.c** as usual, but perform an implicit check-out. The first form also locks the checked in revision, the second one doesn't. Thus, these options save you one check-out operation. The first form is useful if you want to continue editing, the second one if you just want to read the file. Both update the identification markers in your working file (see below).

You can give *ci* the number you want assigned to a checked in revision. Assume all your revisions were numbered 1.1, 1.2, 1.3, etc., and you would like to start release 2. The command

```
ci -r2 f.c
```

or

```
ci -r2.1 f.c"
```

assigns the number 2.1 to the new revision. From then on, *ci* will number the subsequent revisions with 2.2, 2.3, etc. The corresponding *co* commands

```
co -r2 f.c
```

```
co -r2.1 f.c
```

retrieve the latest revision numbered 2.x and the revision 2.1, respectively. *co* without a revision number selects the latest revision on the *trunk*, i.e. the highest revision with a number consisting of two fields. Numbers with more than two fields are needed for branches. For example, to start a branch at revision 1.3, invoke

```
ci -r1.3.1 f.c
```

This command starts a branch numbered 1 at revision 1.3, and assigns the number 1.3.1.1 to the new revision. For more information about branches, see *rcsfile*.

Automatic Identification

RCS can put special strings for identification into your source and object code. To obtain such identification, place the marker **\$Id\$**

into your text, for instance inside a comment. RCS will replace this marker with a string of the form

```
$Id: filename revision date time author state $
```

With such a marker on the first page of each module, you can always see with which revision you are working. RCS keeps the markers up to date automatically. To propagate the markers into your object code, simply put them into literal character strings. In C, this is done as follows:

```
static char rcsid[] = "$Id$";
```

The command *ident* extracts such markers from any file, even object code and dumps. Thus, *ident* lets you find out which revisions of which modules were used in a given program.

You may also find it useful to put the marker **\$Log\$** into your text, inside a comment. This marker accumulates the log messages that are requested during check-in. Thus, you can maintain the complete history of your file directly inside it. There are several additional identification markers; see **co(1)** for details.

IDENTIFICATION

Author: Walter F. Tichy.

Revision Number: 1.4; Release Date: 1997/02/27.

Copyright © 1982, 1988, 1989 by Walter F. Tichy.

Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

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

rcsintro

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

NAME

rcsfile – format of RCS file

DESCRIPTION

An RCS file's contents are described by the grammar below.

The text is free format: space, backspace, tab, newline, vertical tab, form feed, and carriage return (collectively, *white space*) have no significance except in strings. However, an RCS file must end in a newline character.

Strings are enclosed by @. If a string contains a @, it must be doubled; otherwise, strings may contain arbitrary binary data.

The meta syntax uses the following conventions: '|' (bar) separates alternatives; '{' and '}' enclose optional phrases; '{' and '*}' enclose phrases that may be repeated zero or more times; '{' and '}'+' enclose phrases that must appear at least once and may be repeated; Terminal symbols are in **boldface**; nonterminal symbols are in *italics*.

```

rcstext      ::=  admin {delta}* desc {deltatext}*

admin        ::=  head {num};
                  { branch {num}; }
                  access {id}*;
                  symbols {id : num}*;
                  locks {id : num}*; {strict ;}
                  { comment {string}; }
                  { expand {string}; }
                  { newphrase }*

delta        ::=  num
                  date num;
                  author id;
                  state {id};
                  branches {num}*;
                  next {num};
                  { newphrase }*

desc         ::=  desc string

deltatext    ::=  num
                  log string
                  { newphrase }*
                  text string

num          ::=  {digit{.}}+

digit        ::=  0 | 1 | ... | 9

```

```

id ::= letter{idchar}*

letter ::= any letter

idchar ::= any visible graphic character except special

special ::= $ | , | . | : | ; | @

string ::= @{any character, with @ doubled}*@

newphrase ::= id word* ;

word ::= id | num | string | :

```

Identifiers are case sensitive. Keywords are in lower case only. The sets of keywords and identifiers may overlap. In most environments RCS uses the ISO 8859/1 encoding: letters are octal codes 101–132, 141–172, 300–326, 330–366 and 370–377, visible graphic characters are codes 041–176 and 240–377, and white space characters are codes 010–015 and 040.

The *newphrase* productions in the grammar are reserved for future extensions to the format of RCS files. No *newphrase* will begin with any keyword already in use.

The *delta* nodes form a tree. All nodes whose numbers consist of a single pair (e.g., 2.3, 2.1, 1.3, etc.) are on the trunk, and are linked through the **next** field in order of decreasing numbers. The **head** field in the *admin* node points to the head of that sequence (i.e., contains the highest pair). The **branch** node in the *admin* node indicates the default branch (or revision) for most RCS operations. If empty, the default branch is the highest branch on the trunk.

All *delta* nodes whose numbers consist of $2n$ fields ($n \geq 2$) (e.g., 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first $2n-1$ number fields are identical are linked through the **next** field in order of increasing numbers. For each such sequence, the *delta* node whose number is identical to the first $2n-2$ number fields of the deltas on that sequence is called the branchpoint. The **branches** field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

Example:

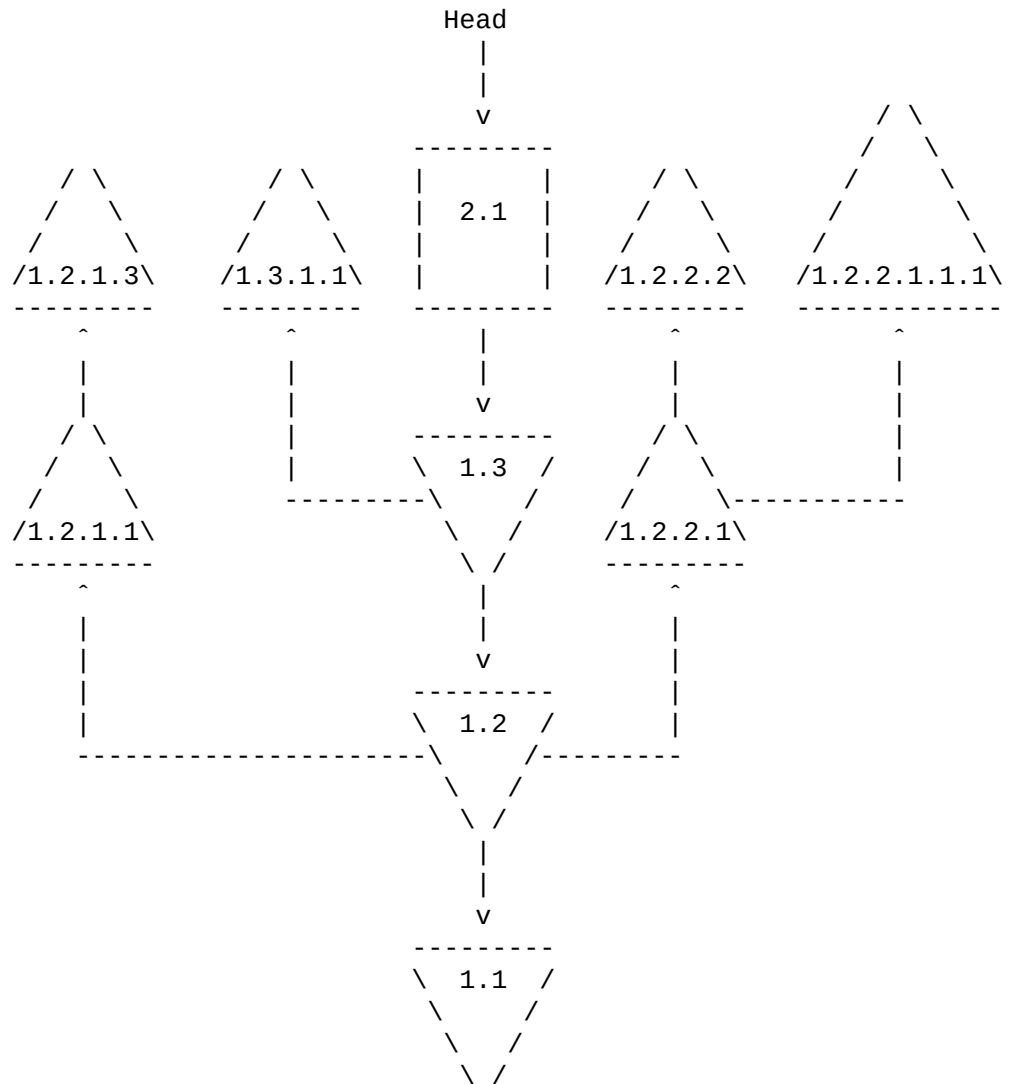


Fig. 1: A revision tree

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.

Revision Number: 1.5; Release Date: 1997/02/27.

Copyright © 1982, 1988, 1989 by Walter F. Tichy.

Copyright © 1990, 1991 by Paul Eggert.

SEE ALSO

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

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

NAME

re_format – POSIX 1003.2 regular expressions

DESCRIPTION

Regular expressions (REs), as defined in POSIX 1003.2, come in two forms: basic REs (roughly those of *ed*) and extended (roughly those of *egrep*). POSIX 1003.2 leaves some aspects of RE syntax and semantics open; ‘†’ marks decisions on these aspects that may not be fully portable to other 1003.2 implementations.

Basic Regular Expressions

A basic regular expression is one or more “pieces” concatenated. It matches a match for the first, followed by a match for the second, etc.

A **piece** is an *atom* possibly followed by a single† ‘*’ or *bound*. An atom followed by ‘*’ matches a sequence of 0 or more matches of the atom.

A **bound** is ‘\{’ followed by an unsigned decimal integer, possibly followed by ‘,’ possibly followed by another unsigned decimal integer, always followed by ‘}’. The integers must lie between 0 and RE_DUP_MAX (255†) inclusive, and if there are two of them, the first may not exceed the second. An atom followed by a bound containing one integer *i* and no comma matches a sequence of exactly *i* matches of the atom. An atom followed by a bound containing one integer *i* and a comma matches a sequence of *i* or more matches of the atom. An atom followed by a bound containing two integers *i* and *j* matches a sequence of *i* through *j* (inclusive) matches of the atom.

An **atom** is a regular expression enclosed in ‘\()’ (matching a match for the regular expression), a *bracket expression* (see below), ‘.’ (matching any single character), ‘^’ (matching the null string at the beginning of a line), ‘\$’ (matching the null string at the end of a line), a ‘\’ followed by one of the characters ‘.[\$*\’ (matching that character taken as an ordinary character), a ‘\’ followed by any other character† (matching that character taken as an ordinary character, as if the ‘\’ had not been present†), a “back reference,” or a single character with no other significance (matching that character). It is illegal to end an RE with ‘\’.

‘^’ is an ordinary character except at the beginning of the RE or† the beginning of a parenthesized subexpression, ‘\$’ is an ordinary character except at the end of the RE or† the end of a parenthesized subexpression, and ‘*’ is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading ‘^’).

A **back reference** is ‘\’ followed by a non-zero decimal digit *d*. It matches the same sequence of characters matched by the *d*th parenthesized subexpression (numbering subexpressions by the positions of their opening parentheses, left to right), so that (e.g.) ‘\([bc]\)1’ matches ‘bb’ or ‘cc’ but not ‘bc’.

Extended Regular Expressions

As the name suggests, extended regular expressions extend the capabilities of basic regular expressions by adding several new constructs and by eliminating back references.

The following new characters are defined for extended regular expressions:

+ ? { } () |

An atom is now is a regular expression enclosed in '()' instead of '\(\)', an empty set of '()' (matching the null string)†, a *bracket expression* (see below), '.' (matching any single character), '^' (matching the null string at the beginning of a line), '\$' (matching the null string at the end of a line), a '\' followed by one of the characters '^.\$()|*+?{\'' (matching that character taken as an ordinary character), a '\' followed by any other character† (matching that character taken as an ordinary character, as if the '\' had not been present†), or a single character with no other significance (matching that character). A '{' followed by a character other than a digit is an ordinary character, not the beginning of a bound†. It is still illegal to end an RE with '\.

An atom followed by '*' matches a sequence of 0 or more matches of the atom. An atom followed by '+' matches a sequence of 1 or more matches of the atom. An atom followed by '?' matches a sequence of 0 or 1 matches of the atom.

The delimiter characters for a *bound* are '{' and '}' instead of '\{' and '\}'.

The '|' character separates *branches*. An extended regular expression matches anything that matches one of the branches. For example, 'alpha|beta' matches either 'alpha' or 'beta'.

Basic Pattern	Extended Pattern	Matches
<i>c</i>	<i>c</i>	Matches the character <i>c</i> except as below
<i>\c</i>	<i>\c</i>	the character <i>c</i>
<i>^</i>	<i>^</i>	the beginning of a line
<i>\$</i>	<i>\$</i>	the end of a line
<i>.</i>	<i>.</i>	any single character
<i>[list]</i>	<i>[list]</i>	any one of the characters in <i>list</i>
<i>[^list]</i>	<i>[^list]</i>	any one of the characters not in <i>list</i>
<i>()</i>	<i>()</i>	the null string
<i>\(a\)</i>	<i>(a)</i>	groups pattern <i>a</i> into a single expression
<i>a*</i>	<i>a*</i>	0 or more occurrences of <i>a</i>
	<i>a+</i>	1 or more occurrences of <i>a</i>
	<i>a?</i>	0 or 1 occurrences of <i>a</i>
<i>a\{n\}</i>	<i>a{n}</i>	<i>n</i> occurrences of <i>a</i>
<i>a\{n, \}</i>	<i>a{n, }</i>	<i>n</i> or more occurrences of <i>a</i>
<i>a\{n, m\}</i>	<i>a{n, m}</i>	between <i>n</i> and <i>m</i> occurrences of <i>a</i>
	<i>a b</i>	pattern <i>a</i> or pattern <i>b</i>
<i>\d</i>		refers back to matched expression <i>d</i>

Bracket Expressions

A **bracket expression** is a list of characters enclosed in '['. It normally matches any single character from the list (but see below). If the list begins with '^', it matches any single character (but see below) *not* from the rest of the list. If two characters in the list are separated by '-', this is shorthand for the full *range* of characters between those two (inclusive) in the collating sequence, e.g. '[0-9]' in ASCII matches any decimal digit. It is illegal† for two ranges to share an endpoint, e.g. 'a-c-e'. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal ']' in the list, make it the first character (following a possible '^'). To include a literal '-', make it the first or last character, or the second endpoint of a range. To use a literal '-' as the first endpoint of a range, enclose it in '[' and ']' to make it a collating element (see below). With the exception of these and some combinations using '[' (see next paragraphs), all other special characters, including '\', lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in '[' and ']' stands for the sequence of characters of that collating element. The sequence is a single element of the bracket expression's list. A bracket expression containing a multi-character collating element can thus match more than one character, e.g. if the collating sequence includes a 'ch' collating element, then the RE '[[.ch.]]*c' matches the first five characters of 'chchcc'.

Within a bracket expression, a collating element enclosed in '[=' and '=]' is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (If there are no other equivalent collating elements, the treatment is as if the enclosing delimiters were '[' and '].') For example, if o and ô are the members of an equivalence class, then '[[=o=]]', '[[=ô=]]', and '[oô]' are all synonymous. An equivalence class may not† be an endpoint of a range.

Within a bracket expression, the name of a **character class** enclosed in '[' and ']' stands for the list of all characters belonging to that class. Standard character class names are:

alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

These stand for the character classes defined in *ctype()*. A locale may provide others. A character class may not be used as an endpoint of a range.

There are two special cases† of bracket expressions: the bracket expressions '[[:<:]]' and '[[:>:]]' match the null string at the beginning and end of a word respectively. A *word* is defined as a sequence of word characters which is neither preceded nor followed by word characters. A word character is an *alnum* character (as defined by *ctype()*) or an underscore. This is an extension, compatible with but not specified by POSIX 1003.2, and should be used with

caution in software intended to be portable to other systems.

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, it matches the longest. Subexpressions also match the longest possible substrings, subject to the constraint that the whole match be as long as possible, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that higher-level subexpressions thus take priority over their lower-level component subexpressions.

Match lengths are measured in characters, not collating elements. A null string is considered longer than no match at all. For example, `bb*` matches the three middle characters of `abbbc`, `(wee | week)(knights | nights)` matches all ten characters of `weeknights`, when `(.*)*` is matched against `abc` the parenthesized subexpression matches all three characters, and when `(a*)*` is matched against `bc` both the whole RE and the parenthesized subexpression match the null string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, e.g. `'x'` becomes `'[xX]'`. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that (e.g.) `'[x]'` becomes `'[xX]'` and `'[^x]'` becomes `'[^xX]'`.

No particular limit is imposed on the length of REs†. Programs intended to be portable should not employ REs longer than 256 bytes, as an implementation can refuse to accept such REs and remain POSIX-compliant.

SEE ALSO

`ctype()`, `regex()`

POSIX 1003.2, section 2.8 (Regular Expression Notation).

BUGS

The current 1003.2 spec says that `'` is an ordinary character in the absence of an unmatched `'`; this was an unintentional result of a wording error, and change is likely. Avoid relying on it.

1003.2's specification of case-independent matching is vague. The "one case implies all cases" definition given above is current consensus among implementors as to the right interpretation.

The syntax for word boundaries is incredibly ugly.

NAME

syslog.conf – syslogd configuration file

DESCRIPTION

The **syslog.conf** file is the configuration file for the *syslogd* program. It consists of lines with two fields: the *selector* field which specifies the types of messages and priorities to which the line applies, and an *action* field which specifies the action to be taken if a message *syslogd* receives matches the selection criteria. The *selector* field is separated from the *action* field by one or more tab characters.

The *selector* function is encoded as a *facility*, a period (“.”), and a *level*, with no intervening white-space. Both the *facility* and the *level* are case insensitive:

```
facility[;facility...].level[,level...]
```

The *facility* describes the part of the system generating the message, and is one of the following keywords: **auth**, **authpriv**, **cron**, **daemon**, **ftp**, **kern**, **lpr**, **mail**, **mark**, **news**, **syslog**, **user**, **uucp** and **local0** through **local7**. These keywords (with the exception of **mark**) correspond to the similar “LOG_” values specified to the *openlog()* and *syslog()* library routines. An asterisk (*) can be used to represent all facilities.

The *level* describes the severity of the message, and is a keyword from the following ordered list (higher to lower): **emerg**, **alert**, **crit**, **err**, **warning**, **notice** and **debug**. These keywords correspond to the similar (LOG_) values specified to the *syslog()* library routine. The special *level* **none** disables a particular *facility*; an asterisk specifies all possible levels.

See *syslog()* for a further descriptions of both the *facility* and *level* keywords and their significance.

If a received message matches the specified *facility* and is of the specified *level* (or a higher the action specified in the *action* field will be taken.

Multiple *selectors* may be specified for a single *action* by separating them with semicolon (“;”) characters. It is important to note, however, that each *selector* can modify the ones preceding it.

Multiple *facilities* may be specified for a single *level* by separating them with comma (“,”) characters.

The special *facility* “mark” receives a message at priority “info” every 20 minutes (see *syslogd*). This is not enabled by a *facility* field containing an asterisk.

The *action* field of each line specifies the action to be taken when the *selector* field selects a message. There are four forms:

- A pathname (beginning with a leading slash). Selected messages are appended to the file. The pathnames are assumed to be relative to **\$OPENNT_ROOT**. That is, the pathname **/var/adm/log/logger** represents the file **\$OPENNT_ROOT/var/adm/log/logger**.
- A hostname (preceded by an at (“@”) sign). Selected messages are forwarded to the *syslogd* program on the named host.

- A comma separated list of users. Selected messages are written to those users if they are logged in.
- An asterisk. Selected messages are written to all logged-in users.

Blank lines and lines whose first non-blank character is a hash (“#”) character are ignored.

EXAMPLES

A configuration file might appear as follows:

```
# Log all kernel messages, authentication messages of
# level notice or higher and anything of level err or
# higher to the console.
# Don't log private authentication messages!
*.err;kern.*;auth.notice;authpriv.none          /dev/console
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none                  /var/log/messages
# The authpriv file has restricted access.
authpriv.*                                       /var/log/secure
# Log all the mail messages in one place.
mail.*                                           /var/log/maillog
# Everybody gets emergency messages,
# plus log them on another machine.
*.emerg                                          *
*.emerg                                          @arpa.berkeley.edu
# Root and Eric get alert and higher messages.
*.alert                                         root,eric
# Save mail and news errors of level err
# and higher in a special file.
uucp,news.crit                                  /var/log/spoolerr
```

FILES**\$OPENNT_ROOT/etc/syslog.conf**

The *syslogd* configuration file.

NOTES

The effects of multiple selectors are sometimes not intuitive. For example “mail.crit,*err” will select “mail” facility messages at the level of “err” or higher, not at the level of “crit” or higher.

SEE ALSO

syslogd

syslog()

NAME

termcap – terminal capability data base

SYNOPSIS

termcap

DESCRIPTION

The **Termcap** file is a data base describing terminals, used, for example, by *vi* and *curses()*. Terminals are described in **termcap** by giving a set of capabilities that they have and by describing how operations are performed. Padding requirements and initialization sequences are included in **termcap**.

Entries in **termcap** consist of a number of ‘:’-separated fields. The first entry for each terminal gives the names that are known for the terminal, separated by ‘|’ characters. The first name given is the most common abbreviation for the terminal. The last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should be in lower case and contain no blanks; the last name may well contain upper case characters and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, thus “hp2621” This name should not contain hyphens. Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Therefore, a “vt100” in 132-column mode would be “vt100-w.” The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-w
-am	With automatic margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-n	Number of lines on screen aaa-60	
-na	No arrow keys (leave them in local)	concept100-na
-np	Number of pages of memory	concept100-4p
-rv	Reverse video	concept100-rv

CAPABILITIES

The characters in the *Notes* function field in the table have the following meanings (more than one may apply to a capability):

- N** indicates numeric parameter(s)
- P** indicates that padding may be specified
- *** indicates that padding may be based on the number of lines affected
- o** indicates capability is obsolete

“Obsolete” capabilities have no **terminfo** equivalents, since they were considered useless, or are subsumed by other capabilities. New software should not rely on them at all.

Name	Type	Notes	Description
ae	str	(P)	End alternate character set.
AL	str	(NP*)	Add <i>n</i> new blank lines
al	str	(P*)	Add new blank line.
am	bool		Terminal has automatic margins.
as	str	(P)	Start alternate character set.
bc	str	(o)	Backspace if not ^H.
bl	str	(P)	Audible signal (bell).
bs	bool	(o)	Terminal can backspace with ^H
bt	str	(P)	Back tab.
bw	bool		le (backspace) wraps from column 0 to last column.
CC	str		Terminal settable command character in prototype.
cd	str	(P*)	Clear to end of display.
ce	str	(P)	Clear to end of line.
ch	str	(NP)	Set cursor column (horizontal position).
cl	str	(P*)	Clear screen and home cursor.
CM	str	(NP)	Memory-relative cursor addressing.
cm	str	(NP)	Screen-relative cursor motion.
co	num		Number of columns in a line (See <i>BUGS</i> section below).
cr	str	(P)	Carriage return.
cs	str	(NP)	Change scrolling region (VT100).
ct	str	(P)	Clear all tab stops.
cv	str	(NP)	Set cursor row (vertical position).
da	bool		Display may be retained above the screen.
dB	num	(o)	Milliseconds of bs delay needed (default 0).
db	bool		Display may be retained below the screen.
DC	str	(NP*)	Delete <i>n</i> characters.
dC	num	(o)	Milliseconds of cr delay needed (default 0).
dc	str	(P*)	Delete character.
dF	num	(o)	Milliseconds of ff delay needed (default 0).
DL	str	(NP*)	Delete <i>n</i> lines.
dl	str	(P*)	Delete line.
dm	str		Enter delete mode.
dN	num	(o)	Milliseconds of nl delay needed (default 0).
DO	str	(NP*)	Move cursor down: <i>n</i> lines.
do	str		Down one line.
Name	Type	Notes	Description
ds	str		Disable status line. Milliseconds of horizontal tab delay

Name	Type	Notes	Description
			needed (default 0).
dT	num	(o)	
dV	num	(o)	Milliseconds of vertical tab delay needed (default 0).
ec	str	(NP)	Erase <i>n</i> characters.
ed	str		End delete mode.
ei	str		End insert mode.
eo	bool		Can erase overstrikes with a blank.
EP	bool	(o)	Even parity.
es	bool		Escape can be used on the status line.
ff	str	(P*)	Hardcopy terminal page eject.
fs	str		Return from status line.
gn	bool		Generic line type, for example dialup, switch).
hc	bool		Hardcopy terminal.
HD	bool	(o)	Half-duplex.
hd	str		Half-line down (forward 1/2 linefeed).
ho	str	(P)	Home cursor.
hs	bool		Has extra "status line."
hu	str		Half-line up (reverse 1/2 linefeed).
hz	bool		Cannot print "" (Hazeltime).
i1-i3	str		Terminal initialization strings (<i>terminfo</i> only)
IC	str	(NP*)	Insert <i>n</i> blank characters.
ic	str	(P*)	Insert character.
if	str		Name of file containing initialization string.
im	str		Enter insert mode.
in	bool		Insert mode distinguishes nulls.
iP	str		Pathname of program for initialization (<i>terminfo</i> only).
ip	str	(P*)	Insert pad after character inserted.
is	str		Terminal initialization string (<i>termcap</i> only).
it	num		Tabs initially every <i>n</i> positions.
K1	str		Sent by keypad upper left.
K2	str		Sent by keypad upper right.
K3	str		Sent by keypad center.
K4	str		Sent by keypad lower left.
K5	str		Sent by keypad lower right.
k0-k9	str		Sent by function keys 0-9.
kA	str		Sent by insert-line key.
ka	str		Sent by clear-all-tabs key.
kb	str		Sent by backspace key.
kC	str		Sent by clear-screen or erase key.
kD	str		Sent by delete-character key.

Name	Type	Notes	Description
kE	str		Sent by clear-to-end-of-line key.
ke	str		Out of “keypad transmit” mode.
kF	str		Sent by scroll-forward / down key.
kH	str		Sent by home-down key.
kh	str		Sent by home key.
kI	str		Sent by insert-character or enter-insert-mode key.
kL	str		Sent by delete-line key.
kl	str		Sent by left-arrow key.
kM	str		Sent by insert key while in insert mode.
km	bool		Has a “meta” key (shift, sets parity bit).
kN	str		Sent by next-page key.
kn	num	(o)	Number of function (k0–k9) keys (default 0).
ko	str	(o)	Termcap entries for other non-function keys.
kP	str		Sent by previous-page key.
kR	str		Sent by scroll-backward / up key.
kr	str		Sent by right-arrow key.
kS	str		Sent by clear-to-end-of-screen key.
ks	str		Put terminal in “keypad transmit” mode.
kT	str		Sent by set-tab key.
kt	str		Sent by clear-tab key.
ku	str		Sent by up-arrow key.
l0-l9	str		Labels on function keys if not “fn”.
LC	bool	(o)	Lower-case only.
LE	str	(NP)	Move cursor left <i>n</i> positions.
le	str	(P)	Move cursor left one position.
li	num		Number of lines on screen or page (See <i>BUGS</i> section below).
ll	str		Last line, first column
lm	num		Lines of memory if > li (0 means varies).
ma	str	(o)	Arrow key map (used by <i>vi</i> version 2 only).
mb	str		Turn on blinking attribute.
md	str		Turn on bold (extra bright) attribute.
me	str		Turn off all attributes.
mh	str		Turn on half-bright attribute.
mi	bool		Safe to move while in insert mode.
mk	str		Turn on blank attribute (characters invisible).
ml	str	(o)	Memory lock on above cursor.

Name	Type	Notes	Description
mm	str		Turn on “meta mode” (8th bit).
mo	str		Turn off “meta mode.”
mp	str		Turn on protected attribute.
mr	str		Turn on reverse-video attribute.
ms	bool		Safe to move in standout modes.
mu	str	(o)	Memory unlock (turn off memory lock).
nc	bool	(o)	No correctly-working cr (Datamedia 2500, Hazeltine 2000).
nd	str		Non-destructive space (cursor right).
Name	Type	Notes	Description
NL	bool	(o)	\n is newline, not line feed.
nl	str	(o)	Newline character if not \n.
ns	bool	(o)	Terminal is a CRT but doesn’t scroll.
nw	str	(P)	Newline (behaves like cr followed by do).
OP	bool	(o)	Odd parity.
os	bool		Terminal overstrikes.
pb	num		Lowest baud where delays are required.
pc	str		Pad character (default NUL).
pf	str		Turn off the printer.
pk	str		Program function key <i>n</i> to type string <i>s</i> (<i>terminfo</i> only).
pl	str		Program function key <i>n</i> to execute string <i>s</i> (<i>terminfo</i> only).
pO	str	(N)	Turn on the printer for <i>n</i> bytes.
po	str		Turn on the printer.
ps	str		Print contents of the screen.
pt	bool	(o)	Has hardware tabs (may need to be set with is).
px	str		Program function key <i>n</i> to transmit string <i>s</i> (<i>terminfo</i> only).
r1-r3	str		Reset terminal completely to sane modes (<i>terminfo</i> only).
rc	str	(P)	Restore cursor to position of last sc .
rf	str		Name of file containing reset codes.
RI	str	(NP)	Move cursor right <i>n</i> positions.
rp	str	(NP*)	Repeat character <i>c</i> <i>n</i> times.
rs	str		Reset terminal completely to sane modes (<i>termcap</i> only).
sa	str	(NP)	Define the video attributes.
sc	str	(P)	Save cursor position.
se	str		End standout mode.
SF	str	(NP*)	Scroll forward <i>n</i> lines.
sf	str	(P)	Scroll text up.
sg	num		Number of garbage chars left by so or se (default 0).

Name	Type	Notes	Description
so	str		Begin standout mode.
SR	str	(NP*)	Scroll backward <i>n</i> lines.
sr	str	(P)	Scroll text down.
st	str		Set a tab in all rows, current column.
ta	str	(P)	Tab to next 8-position hardware tab stop.
tc	str		Entry of similar terminal — must be last.
te	str		String to end programs that use <i>termcap</i> . t} tistrT{ String to begin programs that use <i>termcap</i> .
ts	str	(N)	Go to status line, column <i>n</i>
UC	bool	(o)	Upper-case only.
uc	str		Underscore one character and move past it.
ue	str		End underscore mode.

Name	Type	Notes	Description
ug	num		Number of garbage chars left by us or ue (default 0).
ul	bool		Underline character overstrikes.
UP	str	(NP*)	Move cursor up <i>n</i> lines.
up	str		Upline (cursor up).
us	str		Start underscore mode.
vb	str		Visible bell (must not move cursor).
ve	str		Make cursor appear normal (undo vs / vi).
vi	str		Make cursor invisible.
vs	str		Make cursor very visible.
vt	num		Virtual terminal number (not supported on all systems).
wi	str	(N)	Set current window.
ws	num		Number of columns in status line.
xb	bool		Beehive (f1 =ESC, f2 =^C).
xn	bool		Newline ignored after 80 cols (Concept).
xo	bool		Terminal uses xoff / xon (DC3 / DC1) handshaking.
xr	bool	(o)	Return acts like ce cr nl (Delta Data).
xs	bool		Standout not erased by overwriting (Hewlett-Packard).
xt	bool		Tabs ruin, magic so char (Telera 1061).
xx	bool	(o)	Tektronix 4025 insert-line.

A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the **termcap** file as of this writing.

```
ca | concept100 | c100 | concept | c104 | concept100-4p | HDS Concept-100:\
:al=3*\E^R:am:bl=^G:cd=16*\E^C:ce=16\E^U:cl=2*\L:cm=\Ea%+ %+ :\
:co#80: .cr=9^M:db:dc=16\E^A:dl=3*\E^B:do=^J:ei=\E\200:eo:im=\E^P:in:\
:ip=16*:is=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200\Eo\47\E:k1=\E5:\
:k2=\E6:k3=\E7:kb=^h:kd=\E<:ke=\Ex:kh=\E?:kl=\E>:kr=\E=:ks=\EX:\
```

```
:ku=\E; :le=^H: li#24: mb=\EC: me=\EN\200: mh=\EE: mi: mk=\EH: mp=\EI:\n  
:mr=\ED: nd=\E=: pb#9600: rp=0.2*\Er%.%+ :se=\Ed\Ee: sf=^J: so=\EE\ED:\n  
:. ta=8\t: te=\Ev \200\200\200\200\200\200\Ep\r\n:\n  
:ti=\EU\Ev 8p\Ep\r: ue=\Eg: ul: up=\E; :us=\EG:\n  
:vb=\Ek\200\200\200\200\200\200\200\200\200\200\200\200\200\EK:\n  
:ve=\Ew: vs=\EW: vt#8: xn:\n  
:bs: cr=^M: dC#9: dT#8: nl=^J: ta=^I: pt:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with “#.”

Types of Capabilities

Capabilities in **termcap** are of three types: Boolean capabilities, which indicate particular features that the terminal has; numeric capabilities, giving the size of the display or the size of other attributes; and string capabilities, which give character sequences that can be used to perform particular terminal operations. All capabilities have two-letter codes. For instance, the fact that the Concept has *automatic* margins (an automatic return and linefeed when the end of a line is reached) is indicated by the Boolean capability **am**. Hence the description of the Concept includes **am**.

Numeric capabilities are followed by the character '#' then the value. In the example above **co**, which indicates the number of columns the display has, gives the value '80' for the Concept.

Finally, string-valued capabilities, such as **ce** (clear-to-end-of-line sequence) are given by the two-letter code, an '=', then a string ending at the next following '.'. A delay in milliseconds may appear after the '=' in such a capability, which causes padding characters to be supplied by *tputs* after the remainder of the string is sent to provide this delay. The delay can be either a number, such as '20', or a number followed by an '*', such as '3*'. An '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-line padding required. (In the case of insert-character, the factor is still the number of *lines* affected; this is always 1 unless the terminal has **in** and the software uses it.) When an '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per line to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string-valued capabilities for easy encoding of control characters there. `\E` maps to an ESC character, `^X` maps to a control-X for any appropriate X, and the sequences `\n` `\r` `\t` `\b` `\f` map to linefeed, return, tab, backspace, and formfeed, respectively. Finally, characters may be given as three octal digits after a `\`, and the characters `^` and `\` may be given as `^` and `\\`. If it is necessary to place a `:` in a capability it must be escaped in octal as `\072`. If it is necessary to place a NUL character in a string capability it must be encoded as `\200`. (The routines that deal with **termcap** use C strings and strip the high bits of the output very late, so that a `\200` comes out as a `\000` would.)

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the first **cr** and **ta** in the

example above.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **termcap** and to build up a description gradually, using partial descriptions with *vi* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **termcap** file to describe it or bugs in *vi*. To easily test a new terminal description you are working on you can put it in your home directory in a file called **.termcap** and programs will look there before looking in **/usr/share/misc/termcap**. You can also set the environment variable **TERMPATH** to a list of absolute file pathnames (separated by spaces or colons), one of which contains the description you are working on, and programs will search them in the order listed, and nowhere else. See *termcap()*. The **TERMCAP** environment variable is usually set to the **termcap** entry itself to avoid reading files when starting up a program.

To get the padding for insert-line right (if the terminal manufacturer did not document it), a severe test is to use *vi* to edit **/etc/passwd** at 9600 baud, delete roughly 16 lines from the middle of the screen, then hit the **u** key several times quickly. If the display messes up, more padding is usually needed. A similar test can be used for insert-character.

Basic Capabilities

The number of columns on each line of the display is given by the **co** numeric capability. If the display is a CRT, then the number of lines on the screen is given by the **li** capability. If the display wraps around to the beginning of the next line when the cursor reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, the code to do this is given by the **cl** string capability. If the terminal overstrikes (rather than clearing the position when a character is overwritten), it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as the Tektronix 4010 series, as well as to hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage-return, **^M**.) If there is a code to produce an audible signal (bell, beep, etc.), give this as **bl**.

If there is a code (such as backspace) to move the cursor one position to the left, that capability should be given as **le**. Similarly, codes to move to the right, up, and down should be given as **nd**, **up**, and **do**, respectively. These *local cursor motions* should not alter the text they pass over; for example, you would not normally use “nd= ” unless the terminal has the **os** capability, because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **termcap** have undefined behavior at the left and top edges of a display. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up off the top using local cursor motions.

In order to scroll text up, a program goes to the bottom left corner of the screen and sends the **sf** (index) string. To scroll text down, a program goes to the top

left corner of the screen and sends the **sr** (reverse index) string. The strings **sf** and **sr** have undefined behavior when not on their respective corners of the screen. Parameterized versions of the scrolling sequences are **SF** and **SR**, which have the same semantics as **sf** and **sr** except that they take one parameter and scroll that many lines. They also have undefined behavior except at the appropriate corner of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output there, but this does not necessarily apply to **nd** from the last column. Leftward local motion is defined from the left edge only when **bw** is given; then an **le** from the left edge will move to the right edge of the previous row. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the **termcap** description usually assumes that this feature is on, *i.e.*, **am**. If the terminal has a command that moves to the first column of the next line, that command can be given as **nw** (newline). It is permissible for this to clear the remainder of the current line, so if the terminal has no correctly-working CR and LF it may still be possible to craft a working **nw** out of one or both of them.

These capabilities suffice to describe hardcopy and “glass-tty” terminals. Thus the Teletype model 33 is described as

```
T3 | tty33 | 33 | tty | Teletype model 33:\
      :bl=^G:co#72:cr=^M:do=^J:hc:os:
```

and the Lear Siegler ADM-3 is described as

```
l3 | adm3 | 3 | LSI ADM-3:\
:am:bl=^G:cl=^Z:co#80:cr=^M:do=^J:le=^H:li#24:sf=^J:
```

Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with *printf(3)*-like escapes **%x** in it, while other characters are passed through unchanged. For example, to address the cursor the **cm** capability is given, using two parameters: the row and column to move to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory. If the terminal has memory-relative cursor addressing, that can be indicated by an analogous **CM** capability.)

The **%** encodings have the following meanings:

%%	output ‘%’
%d	output value as in <i>printf</i> %d
%2	output value as in <i>printf</i> %2d
%3	output value as in <i>printf</i> %3d
%.	output value as in <i>printf</i> %c
%+x	add x to value, then do %.
%>xy	if value > x, then add y, no output
%r	reverse order of two parameters, no output
%i	increment by one, no output
%n	exclusive-or all parameters with 0140 (Datamedia 2500)
%B	BCD (16*(value / 10)) + (value%10), no output
%D	Reverse coding (value – 2*(value%16)), no output (Delta Data).

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent “\E&a12c03Y” padded for 6 milliseconds. Note that the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its **cm** capability is “cm=6\E&%r%2c%2Y.”

The Datamedia 2500 needs the current row and column sent encoded in binary using “%..” Terminals that use “%.” need to be able to backspace the cursor (**le**) and to move the cursor up one line on the screen (**up**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (Programs using **termcap** must set terminal modes so that tabs are not expanded, so **\t** is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM-3A, which offsets row and column by a blank character, thus “cm=\E=%+ %+.”

Row or column absolute cursor addressing can be given as single parameter capabilities **ch** (horizontal position absolute) and **cv** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cm**. If there are parameterized local motions move *n* positions to the right) these can be given as **DO**, **LE**, **RI**, and **UP** with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have **cm**, such as the Tektronix 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to the very upper left corner of the screen), this can be given as **ho**. Similarly, a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **up** from the home position, but a program should never do this itself (unless **ll** does), because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as cursor address (0,0): to the top left corner of the screen, not of memory. (Therefore, the “\EH” sequence on Hewlett-Packard terminals cannot be used for **ho**.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, this should be given as **cd**. **cd** must only be invoked from the first column of a line. (Therefore, it can be simulated by a request to delete a large number of lines, if a true **cd** is not available.)

Insert/Delete Line

If the terminal can open a new blank line before the line containing the cursor, this should be given as **al**; this must be invoked only from the first position of a line. The cursor must then appear at the left of the newly blank line. If the terminal can delete the line that the cursor is on, this should be given as **dl**; this must only be used from the first position on the line to be deleted. Versions of **al** and **dl** which take a single parameter and insert or delete that many lines can be given as **AL** and **DL**. If the terminal has a settable scrolling region (like the VT100), the command to set this can be described with the **cs** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command — the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **sr** or **sf** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory which all commands affect, it should be given as the parameterized string **wi**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order. (This *terminfo* capability is described for completeness. It is unlikely that any **termcap**-using program will support it.)

If the terminal can retain display memory above the screen, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **sr** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character that can be described using **termcap**. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept-100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen then typing text separated by cursor motions. Type “abc def” using local cursor motions (not spaces) between the “abc” and the “def.” Then position the cursor before the “abc” and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall

off the end, then your terminal does not distinguish between blanks and untyped positions. If the “abc” shifts over to the “def” which then move together around the end of the current line and onto the next as you insert, then you have the second type of terminal and should give the capability **in**, which stands for “insert null.” While these are two logically separate attributes (one line vs. multi-line insert mode, and special treatment of untyped spaces), we have seen no terminals whose insert mode cannot be described with the single attribute.

Termcap can describe both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode. Give as **ei** the sequence to leave insert mode. Now give as **ic** any sequence that needs to be sent just before each character to be inserted. Most terminals with a true insert mode will not give **ic**; terminals that use a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ic**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after insertion of a single character can also be given in **ip**. If your terminal needs to be placed into an ‘insert mode’ and needs a special code preceding each inserted character, then both **im/ei** and **ic** can be given, and both will be used. The **IC** capability, with one parameter *n*, will repeat the effects of **ic** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia’s) must not have **mi** because of the way their insert mode works.

Finally, you can specify **dc** to delete a single character, **DC** with one parameter *n* to delete *n* characters, and delete mode by giving **dm** and **ed** to enter and exit delete mode (which is any mode the terminal needs to be placed in for **dc** to work).

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout* mode, representing a good high-contrast, easy-on-the-eyes format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **so** and **se**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces or garbage characters on the screen, as the TVI 912 and Teleray 1061 do, then **sg** should be given to tell how many characters are left.

Codes to begin underlining and end underlining can be given as **us** and **ue**, respectively. Underline mode change garbage is specified by **ug**, similar to **sg**. If the terminal has a code to underline the current character and move the cursor one position to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **mb** (blinking), **md** (bold or extra bright), **mh** (dim or half-bright), **mk** (blanking or invisible text), **mp** (protected), **mr** (reverse video), **me** (turn off *all* attribute modes), **as** (enter alternate character set mode), and **ae** (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of mode, this should be given as **sa** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attributes is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, and alternate character set. Not all modes need be supported by **sa**, only those for which corresponding attribute commands exist. (It is unlikely that a **termcap**-using program will support this capability, which is defined for compatibility with *terminfo*.)

Terminals with the “magic cookie” glitches (**sg** and **ug**), rather than maintaining extra attribute bits for each character cell, instead deposit special “cookies,” or “garbage characters,” when they receive mode-setting sequences, which affect the display algorithm.

Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or when the cursor is addressed. Programs using standout mode should exit standout mode on such terminals before moving the cursor or sending a newline. On terminals where this is not a problem, the **ms** capability should be present to say that this overhead is unnecessary.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as **vb**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to change, for example, a non-blinking underline into an easier-to-find block or blinking underline), give this sequence as **vs**. If there is a way to make the cursor completely invisible, give that as **vi**. The capability **ve**, which undoes the effects of both of these modes, should also be given.

If your terminal correctly displays underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local mode (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left-arrow, right-arrow, up-arrow, down-arrow, and home keys can be given as **kl**, **kr**, **ku**, **kd**, and **kh**, respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0**, **k1**, **k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0**, **l1**, **l9**. The codes transmitted by certain other special keys can be given: **kH** (home down), **kb** (backspace), **ka** (clear all tabs), **kt** (clear the tab stop in this column), **kC** (clear screen or erase), **kD**

(delete character), **kL** (delete line), **kM** (exit insert mode), **kE** (clear to end of line), **kS** (clear to end of screen), **kI** (insert character or enter insert mode), **kA** (insert line), **kN** (next page), **kP** (previous page), **kF** (scroll forward / down), **kR** (scroll backward / up), and **kT** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, then the other five keys can be given as **K1**, **K2**, **K3**, **K4**, and **K5**. These keys are useful when the effects of a 3 by 3 directional pad are needed. The obsolete **ko** capability formerly used to describe “other” function keys has been completely supplanted by the above capabilities.

The **ma** entry is also used to indicate arrow keys on terminals that have single-character arrow keys. It is obsolete but still in use in version 2 of **vi** which must be run on some minicomputers due to memory limitations. This field is redundant with **kl**, **kr**, **ku**, **kd**, and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, and the second character is the corresponding **vi** command. These commands are *h* for **kl**, *j* for **kd**, *k* for **ku**, *l* for **kr**, and *H* for **kh**. For example, the Mime would have “ma=^Hh^Kj^Zk^Xl” indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Tabs and Initialization

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory-relative cursor addressing and not screen-relative cursor addressing, a screen-sized window must be fixed into the display for cursor addressing to work properly. This is also used for the Tektronix 4025, where **ti** sets the command character to be the one used by **termcap**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **termcap** description. They are normally sent to the terminal by the *tset* program each time the user logs in. They will be printed in the following order: **is**; setting tabs using **ct** and **st**; and finally **if**. (uses **i1-i2** instead of **is** and runs the program **iP** and prints **i3** after the other initializations.) A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs** and **rf**. (These strings are output by the *reset* program, which is used when the terminal gets into a wedged state. uses **r1-r3** instead of **rs**.) Commands are normally placed in **rs** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the VT100 into 80-column mode would normally be part of **is**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80-column mode.

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ta** (usually ^I). A “backtab” command which moves leftward to the previous tab stop can be given as **bt**. By convention, if the terminal driver modes indicate that tab stops are being expanded by the computer rather than being sent to the terminal, programs should not use **ta** or **bt** even if they are present, since the user may not have the tab stops properly set. If the terminal

has hardware tabs that are initially set every *n* positions when the terminal is powered up, then the numeric parameter **it** is given, showing the number of positions between tab stops. This is normally used by the *tset* command to determine whether to set the driver mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the **termcap** description can assume that they are properly set.

If there are commands to set and clear tab stops, they can be given as **ct** (clear all tab stops) and **st** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is** or **if**.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the *tset* program to set terminal driver modes appropriately. Delays embedded in the capabilities **cr**, **sf**, **le**, **ff**, and **ta** will cause the appropriate delay bits to be set in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**. For BSD 4.2 *tset*, the delays are given as numeric capabilities **dC**, **dN**, **dB**, **dF**, and **dT** instead.

Miscellaneous

If the terminal requires other than a NUL (zero) character as a pad, this can be given as **pc**. Only the first character of the **pc** string is used.

If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**.

If the terminal has an extra “status line” that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, then the capability **hs** should be given. Special strings to go to a position in the status line and to return from the status line can be given as **ts** and **fs**. **fs** must leave the cursor position in the same place that it was before **ts**. If necessary, the **sc** and **rc** strings can be included in **ts** and **fs** to get this effect.) The capability **ts** takes one parameter, which is the column number of the status line to which the cursor is to be moved. If escape sequences and other special commands such as tab work while in the status line, the flag **es** can be given. A string that turns off the status line (or otherwise erases its contents) should be given as **ds**. The status line is normally assumed to be the same width as the rest of the screen, *i.e.*, **co**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), then its width in columns can be indicated with the numeric parameter **ws**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually **^L**).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters), this can be indicated with the parameterized string **rp**. The first parameter is the character to be repeated and the second is the number of times to repeat it. (This is a

terminfo feature that is unlikely to be supported by a program that uses **termcap**.)

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **CC**. A prototype command character is chosen which is used in all capabilities. This character is given in the **CC** capability to identify it. The following convention is supported on some systems: The environment is to be searched for a *CC* variable, and if found, all occurrences of the prototype character are replaced by the character in the environment variable. This use of the *CC* environment variable is a very bad idea, as it conflicts with *make*.

Terminal descriptions that do not represent a specific kind of known terminal, such as **switch**, **dialup**, **patch**, and **network**, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.)

If the terminal uses xoff/xon (DC3/DC1) handshaking for flow control, give **xo**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted.

If the terminal has a “meta key” which acts as a shift key, setting the 8th bit of any character transmitted, then this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this “meta mode” on and off, they can be given as **mm** and **mo**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. An explicit value of 0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the system virtual terminal protocol, the terminal number can be given as **vt**.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **ps**: print the contents of the screen; **pf**: turn off the printer; and **po**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **pO** takes one parameter and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **pf**, is transparently passed to the printer while **pO** is in effect.

Strings to program function keys can be given as **pk**, **pl**, and **px**. Each of these strings takes two parameters: the function key number to program (from 0 to 9) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The differences among the capabilities are that **pk** causes pressing the given key to be the same as the user typing the given string; **pl** causes the string to be executed by the terminal in local mode; and **px** causes the string to be transmitted to the computer. Unfortunately, due to lack of a definition for string parameters in **termcap**, only *terminfo* supports these capabilities.

Glitches and Braindamage

Hazeltine terminals, which do not allow “ characters to be displayed, should indicate **hz**.

The **nc** capability, now obsolete, formerly indicated Datamedia terminals, which echo `\r\n` for carriage return then ignore a following linefeed.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept, should indicate **xn**.

If **ce** is required to get rid of standout (instead of merely writing normal text on top of it), **xs** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a magic cookie, and that to erase standout mode it is necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the ESC or `^C` characters, has **xb**, indicating that the “f1” key is used for ESC and “f2” for `^C`. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form **xx**.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last*, and the combined length of the entries must not exceed 1024. The capabilities given before **tc** override those in the terminal type invoked by **tc**. A capability can be canceled by placing **xx@** to the left of the **tc** invocation, where **xx** is the capability. For example, the entry

```
hn | 2621-nl:ks@:ke@:tc=2621:
```

defines a “2621-nl” that does not have the **ks** or **ke** capabilities, hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES**\$OPENNT_ROOT/usr/share/misc/termcap**

File containing terminal descriptions. The actual directory depends upon where INTERIX is installed.

\$OPENNT_ROOT/usr/share/misc/termcap.db

Hash database file containing terminal descriptions. The actual directory depends upon where INTERIX is installed.

CAVEATS AND BUGS

The *Note*: **termcap** functions were replaced by **terminfo** in AT&T System V Release 2.0. The transition will be relatively painless if capabilities flagged as “obsolete” are avoided.

Lines and columns are now stored by the kernel as well as in the termcap entry. Most programs now use the kernel information primarily; the information in

this file is used only if the kernel does not have any information.

Vi allows only 256 characters for string capabilities, and the routines in *termlib* do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

Not all programs support all entries.

SEE ALSO

ex, *more*, *tset*, *ul*, *vi*

curses(), *printf()*, *tgetent()*, *tgetnum()*, *tgetflag()*, *tgetstr()*, *tgoto()*

NAME

terminal – the INTERIX terminal window

DESCRIPTION

The INTERIX terminal window (named in the *terminfo* database as *interix* and variations; older versions were named *opennt*) has many of the capabilities of an *xterm* window, although the actual terminal descriptions are closer to a vt100 terminal.

The definitive description of the window is the *terminfo* database description. The *termcap* description is not as complete (it lacks some of the function keys, for example).

A user at the console has some additional capabilities, included below. A console terminal window has a pathname of **/dev/tty***.

Baud Rate

The value for baud rate returned by *stty* is meaningless, since there is no serial connection. Like the *xterm*, the value returned by *stty* is 9600.

Function Keys

The terminal description handles 64 function keys. These are:

F1–F12	The standard function keys.
F13–F24	The function keys in combination with the Shift key.
F25–F36	The function keys in combination with the Alt key.
F37–F48	The function keys in combination with the Control key.
F49–F60	The function keys in combination with the Shift-Control key-pair.
F61–F64	The arrow keys in combination with the Shift key. Note that these are <i>not</i> the arrow keys on the numeric keypad.

For a user at the console, terminals on the screen can be scrolled using the keyboard:

Control-<uparrow>	Scroll one line up.
Control-<dnarrow>	Scroll one line down.
Shift-<PgUp>	Scroll one screen up.
Shift-<PgDn>	Scroll one screen down.

The console beeps when it can scroll no farther.

Window Resizing

The default window size is set using the Windows NT properties for the desktop shortcut, icon, or window. (See the *INTERIX Commands and Utilities User's Guide* for instructions on doing this.)

Any console window on the screen can be resized with the mouse.

When the window is resized, the window sends a SIGWINCH signal to the current application.

Color

Colors can be set using the Windows NT properties for the desktop shortcut, icon, or window. While colors can be set for an open window, they will not affect all utilities until the next time a window is opened.

FILES

The termcap and terminfo databases contain the definitive information on key output codes.

\$OPENNT_ROOT/usr/share/termcap

The termcap database.

\$OPENNT_ROOT/usr/share/terminfo/

The terminfo database.

SEE ALSO

terminfo

NAME

terminfo – terminal capability database

SYNOPSIS

terminfo /*/*

DESCRIPTION

Terminfo is a database describing terminals, used by screen-oriented programs such as *nvi*, *rogue* and libraries such as **curses**. *Terminfo* describes terminals by giving a set of capabilities which they have, by specifying how to perform screen operations, and by specifying padding requirements and initialization sequences.

Entries in *terminfo* consist of a sequence of ‘,’ separated fields (embedded commas may be escaped with a backslash or notated as \072). White space after the ‘,’ separator is ignored. The first entry for each terminal gives the names which are known for the terminal, separated by ‘|’ characters. The first name given is the most common abbreviation for the terminal, the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name, thus “hp2621”. This name should not contain hyphens. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and a mode suffix. Thus, a vt100 in 132 column mode would be vt100-w. The following suffixes should be used where possible:

Suffix	Meaning	Example
-am	With automargins (usually the default)	vt100-am
-m	Mono mode; suppress color	ansi-m
-mc	Magic cookie; spaces when highlighting	wy30-mc
-na	No arrow keys (leave them in local)	c100-na
-nam	Without automatic margins	vt100-nam
-nl	No status line	att4415-nl
-nn	Number of lines on the screen	aaa-60
-np	Number of pages of memory	c100-4p
-ns	No status line	hp2626-ns
-rv	Reverse video	c100-rv
-s	Enable status line	vt100-s
-vb	Use visible bell instead of beep	wy370-vb
-w	Wide mode (> 80 columns, usually 132)	vt100-w

Capabilities

The following is a complete table of the capabilities included in a terminfo description block and available to terminfo-using code. In each line of the table,

The *variable* is the name by which the programmer (at the terminfo level) accesses the capability.

The *capname* is the short name used in the text of the database, and is used by a person updating the database. Whenever possible, capnames are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

The termcap code is the old **termcap** capability name (some capabilities are new, and have names which termcap did not originate).

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short and to allow the tabs in the source file **Caps** to line up nicely.

Finally, the description field attempts to convey the semantics of the capability. You may find some codes in the description field:

- (P) indicates that padding may be specified
- #[1-9] in the description field indicates that the string is passed through tparm with parms as given (#i).
- (P*) indicates that padding may vary in proportion to the number of lines affected
- (#.i) indicates the i^{th} parameter.

These are the boolean capabilities:

Variable	Name	Termcap	Description
<i>auto_left_margin</i>	bw	bw	cub1 wraps from column 0 to last column
<i>auto_right_margin</i>	am	am	terminal has automatic margins
<i>no_esc_ctlc</i>	xs_b	xb	beehive (f1=escape, f2=ctrl C)

Variable	Name	Termcap	Description
<i>ceol_standout_glitch</i>	xhp	xs	standout not erased by overwriting (hp)
<i>eat_newline_glitch</i>	xenl	xn	newline ignored after 80 cols (concept)
<i>erase_overstrike</i>	eo	eo	can erase overstrikes with a blank
<i>generic_type</i>	gn	gn	generic line type
<i>hard_copy</i>	hc	hc	hardcopy terminal
<i>has_meta_key</i>	km	km	Has a meta key, sets

<i>has_status_line</i>	hs	hs	msb high
<i>insert_null_glitch</i>	in	in	has extra status line
<i>memory_above</i>	da	da	insert mode
<i>memory_below</i>	db	db	distinguishes nulls
<i>move_insert_mode</i>	mir	mi	display may be retained
<i>move_standout_mode</i>	msgr	ms	above the screen
<i>over_strike</i>	os	os	display may be retained
<i>status_line_esc_ok</i>	eslok	es	below the screen
<i>dest_tabs_magic_smo</i>	xt	xt	safe to move while in
<i>tilde_glitch</i>	hz	hz	insert mode
<i>transparent_underline</i>	ul	ul	safe to move while in
<i>xon_xoff</i>	xon	xo	standout mode
<i>needs_xon_xoff</i>	nxon	nx	terminal can overstrike
<i>prtr_silent</i>	mc5i	5i	escape can be used on the
<i>hard_cursor</i>	chts	HC	status line
<i>non_rev_rmcup</i>	nrrmc	NR	tabs destructive,
<i>no_pad_char</i>	npc	NP	magic so char (t1061)
<i>non_dest_scroll_region</i>	ndscr	ND	can't print ~ (hazeltine)
<i>can_change</i>	ccc	cc	underline character
<i>back_color_erase</i>	bce	ut	overstrikes
<i>col_addr_glitch</i>	xhpa	YA	terminal uses xon /
Variable	Name	Termcap	Description
<i>cr_cancels_micro_mode</i>	crxm	YB	xoff handshaking
<i>has_print_wheel</i>	daisy	YC	padding won't work,
<i>row_addr_glitch</i>	xvpa	YD	xon / xoff required
			printer won't echo
			on screen
			cursor is hard to see
			smcup does not
			reverse rmcup
			pad character does
			not exist
			scrolling region is
			non-destructive
			terminal can re-define
			existing colors
			screen erased with
			background color
			only positive motion
			for hpa / mhpa caps

<i>semi_auto_right_margin</i>	sam	YE	column causes cr
<i>cpi_changes_res</i>	cpix	YF	changing character pitch changes resolution
<i>hue_lightness_saturation</i>	hls	hl	terminal uses only HLS color notation (tektronix)
<i>lpi_changes_res</i>	lpix	YG	changing line pitch changes resolution

This capability is described as a boolean in the XSI Curses standard, but is an (undocumented) string capability in SVr4. It is a boolean here.

Variable	Name	Termcap	Description
<i>get_mouse</i>	getm	Gm	Curses should get button events

These are the numeric capabilities:

Variable	Name	Termcap	Description
<i>columns</i>	cols	co	number of columns in a line
<i>init_tabs</i>	it	it	tabs initially every # spaces
<i>lines</i>	lines	li	number of lines on screen or page
<i>lines_of_memory</i>	lm	lm	lines of memory if > line. 0 => varies
<i>magic_cookie_glitch</i>	xmc	sg	number of blank chars left by sms or rms
<i>padding_baud_rate</i>	pb	pb	lowest baud rate where padding needed
<i>virtual_terminal</i>	vt	vt	virtual terminal number
<i>width_status_line</i>	wsl	ws	columns in status line
<i>num_labels</i>	nlab	Nl	number of labels on screen
<i>label_height</i>	lh	lh	rows in each label
<i>label_width</i>	lw	lw	columns in each label
Variable	Name	Termcap	Description
<i>max_attributes</i>	ma	ma	maximum combined attributes terminal can handle
<i>maximum_windows</i>	wnum	MW	maximum number of defineable windows
<i>max_colors</i>	colors	Co	maximum numbers of colors on screen

<i>max_pairs</i>	pairs	pa	maximum number of color-pairs on screen
<i>no_color_video</i>	ncv	NC	video attributes that can't be used with colors

The following numeric capabilities are present in the SVr4.0 **term** structure, but are not yet documented in the man page. They came in with SVr4's printer support.

Variable	Name	Termcap	Description
<i>buffer_capacity</i>	bufsz	Ya	numbers of bytes buffered before printing
<i>dot_vert_spacing</i>	spinv	Yb	spacing of pins vertically in pins per inch
<i>dot_horz_spacing</i>	spinh	Yc	spacing of dots horizontally in dots per inch
<i>max_micro_address</i>	maddr	Yd	maximum value in micro_..._address
<i>max_micro_jump</i>	mjump	Ye	maximum value in parm_..._micro
<i>micro_char_size</i>	mcs	Yf	character size when in micro mode
<i>micro_line_size</i>	mls	Yg	line size when in micro mode
<i>number_of_pins</i>	npins	Yh	numbers of pins in print-head
<i>output_res_char</i>	orc	Yi	horizontal resolution in units per line
<i>output_res_line</i>	orl	Yj	vertical resolution in units per line
<i>output_res_horz_inch</i>	orhi	Yk	horizontal resolution in units per inch
<i>output_res_vert_inch</i>	orvi	Yl	vertical resolution in units per inch
<i>print_rate</i>	cps	Ym	print rate in chars per second
<i>wide_char_size</i>	widcs	Yn	character step size when in double wide mode

Variable	Name	Termcap	Description
<i>buttons</i>	btns	BT	number of buttons on mouse
<i>bit_image_entwining</i>	bitwin	Yo	number of passed for each bit-image row
<i>bit_image_type</i>	bitype	Yp	type of bit-image device

These are the string capabilities:

Variable	Name	Termcap	Description
-----------------	-------------	----------------	--------------------

<i>back_tab</i>	cbt	bt	back tab (P)
<i>bell</i>	bel	bl	audible signal (bell) (P)
<i>carriage_return</i>	cr	cr	carriage return (P*)
<i>change_scroll_region</i>	csr	cs	change region to line #1 to line #2 (P)
<i>clear_all_tabs</i>	tbc	ct	clear all tab stops (P)
<i>clear_screen</i>	clear	cl	clear screen and home cursor (P*)
<i>column_address</i>	hpa	ch	horizontal position #1, absolute (P)
<i>command_character</i>	cmdch	CC	terminal settable cmd character in prototype !?
<i>cursor_address</i>	cup	cm	move to row #1 columns #2
<i>cursor_down</i>	cud1	do	down one line
<i>cursor_home</i>	home	ho	home cursor (if no cup)
<i>cursor_invisible</i>	civis	vi	make cursor invisible
<i>cursor_left</i>	cub1	le	move left one space
<i>cursor_mem_address</i>	mrcup	CM	memory relative cursor addressing
<i>cursor_normal</i>	cnorm	ve	make cursor appear normal (undo civis / cvvis)
<i>cursor_right</i>	cuf1	nd	move right one space
<i>cursor_to_ll</i>	ll	ll	last line, first column (if no cup)
<i>cursor_up</i>	cuu1	up	up one line
<i>cursor_visible</i>	cvvis	vs	make cursor very visible
<i>delete_character</i>	dch1	dc	delete character (P*)
<i>delete_line</i>	dl1	dl	delete line (P*)
<i>dis_status_line</i>	dsl	ds	disable status line

Variable	Name	Termcap	Description
<i>down_half_line</i>	hd	hd	half a line down
<i>enter_alt_charset_mode</i>	smacs	as	start alternate character set (P)
<i>enter_blink_mode</i>	blink	mb	turn on blinking
<i>enter_bold_mode</i>	bold	md	turn on bold (extra bright) mode
<i>enter_ca_mode</i>	smcup	ti	string to start programs using cup
<i>enter_delete_mode</i>	smdc	dm	enter delete mode
<i>enter_dim_mode</i>	dim	mh	turn on half- bright mode

<i>enter_insert_mode</i>	smir	im	enter insert mode
<i>enter_secure_mode</i>	invis	mk	turn on blank mode (characters invisible)
<i>enter_protected_mode</i>	prot	mp	turn on protected mode
<i>enter_reverse_mode</i>	rev	mr	turn on reverse video mode
<i>enter_standout_mode</i>	sms0	so	begin standout mode
<i>enter_underline_mode</i>	smul	us	begin underline mode
<i>erase_chars</i>	ech	ec	erase #1 characters (P)
<i>exit_alt_charset_mode</i>	rmacs	ae	end alternate character set (P)
<i>exit_attribute_mode</i>	sgr0	me	turn off all attributes
<i>exit_ca_mode</i>	rmcup	te	strings to end pro- grams using cup
<i>exit_delete_mode</i>	rmde	ed	end delete mode
<i>exit_insert_mode</i>	rmir	ei	exit insert mode
<i>exit_standout_mode</i>	rmso	se	exit standout mode
<i>exit_underline_mode</i>	rmul	ue	exit underline mode
<i>flash_screen</i>	flash	vb	visible bell (may not move cursor)
<i>form_feed</i>	ff	ff	hardcopy terminal page eject (P*)
<i>from_status_line</i>	fsl	fs	return from status line
<i>init_1string</i>	is1	i1	initialization string
<i>init_2string</i>	is2	is	initialization string
<i>init_3string</i>	is3	i3	initialization string
<i>init_file</i>	if	if	name of initial- ization file
<i>insert_character</i>	ich1	ic	insert character (P)
<i>insert_line</i>	il1	al	insert line (P*)

Variable	Name	Termcap	Description
<i>insert_padding</i>	ip	ip	insert padding after inserted character
<i>key_backspace</i>	kbs	kb	backspace key
<i>key_catab</i>	ktbc	ka	clear-all-tabs key
<i>key_clear</i>	kclr	kC	clear-screen or erase key
<i>key_ctab</i>	kctab	kt	clear-tab key
<i>key_dc</i>	kdch1	kD	delete-character key
<i>key_dl</i>	kdl1	kL	delete-line key
<i>key_down</i>	kcud1	kd	down-arrow key
<i>key_eic</i>	krmir	kM	sent by rmir or smir in insert mode

<i>key_eol</i>	kel	kE	clear-to-end-of-line key
<i>key_eos</i>	ked	kS	clear-to-end-of-screen key
<i>key_f[0-9]</i>	kf[0-9]	k[0-9]	function keys F0 through f9
<i>key_f10</i>	kf10	k;	F10 function key
<i>key_home</i>	khome	kh	home key
<i>key_ic</i>	kich1	kI	insert-character key
<i>key_il</i>	kil1	kA	insert-line key
<i>key_left</i>	kcub1	kl	left-arrow key
<i>key_ll</i>	kll	kH	last-line key
<i>key_npage</i>	knp	kN	next-page key
<i>key_ppage</i>	kpp	kP	prev-page key
<i>key_right</i>	kcuf1	kr	right-arrow key
<i>key_sf</i>	kind	kF	scroll-forward key
<i>key_sr</i>	kri	kR	scroll-backward key
<i>key_stab</i>	khts	kT	set-tab key
<i>key_up</i>	kcuu1	ku	up-arrow key
<i>keypad_local</i>	rmkx	ke	leave 'keyboard_transmit' mode
<i>keypad_xmit</i>	smkx	ks	enter 'keyboard_transmit' mode
<i>lab_fn</i>	lfn	ln	label on function key <i>fn</i> if not <i>fn</i> ; <i>n</i> is a value from 0-10
<i>meta_off</i>	rmm	mo	turn off meta mode
<i>meta_on</i>	smm	mm	turn on meta mode (8th-bit on)
<i>newline</i>	nel	nw	newline (behave like cr followed by lf)
<i>pad_char</i>	pad	pc	padding char (instead of null)

Variable	Name	Termcap	Description
<i>parm_dch</i>	dch	DC	delete #1 chars (P*)
<i>parm_delete_line</i>	dl	DL	delete #1 lines (P*)
<i>parm_down_cursor</i>	cud	DO	down #1 lines (P*)
<i>parm_ich</i>	ich	IC	insert #1 chars (P*)
<i>parm_index</i>	indn	SF	scroll forward #1 lines (P)
<i>parm_insert_line</i>	il	AL	insert #1 lines (P*)
<i>parm_left_cursor</i>	cub	LE	move #1 chars to the left (P)
<i>parm_right_cursor</i>	cuf	RI	move #1 chars to the right (P*)
<i>parm_rindex</i>	rin	SR	scroll back #1

<i>parm_up_cursor</i>	cuu	UP	lines (P)
<i>pkey_key</i>	pfkey	pk	up #1 lines (P*)
			program function
			key #1 to type
			string #2
<i>pkey_local</i>	pfloc	pl	program function
			key #1 to execute
			string #2
<i>pkey_xmit</i>	px	px	program function
			key #1 to transmit
			string #2
<i>print_screen</i>	mc0	ps	print contents of
			screen
<i>prtr_off</i>	mc4	pf	turn off printer
<i>prtr_on</i>	mc5	po	turn on printer
<i>repeat_char</i>	rep	rp	repeat char #1 #2
			times (P*)
<i>reset_1string</i>	rs1	r1	reset string
<i>reset_2string</i>	rs2	r2	reset string
<i>reset_3string</i>	rs3	r3	reset string
<i>reset_file</i>	rf	rf	name of reset file
<i>restore_cursor</i>	rc	rc	restore cursor to
			last position of sc
<i>row_address</i>	vpa	cv	vertical position
			#1 absolute (P)
<i>save_cursor</i>	sc	sc	save current cursor
			position (P)
<i>scroll_forward</i>	ind	sf	scroll text up (P)
<i>scroll_reverse</i>	ri	sr	scroll text down (P)
<i>set_attributes</i>	sgr	sa	define video
			attributes #1-#9
			(PG9)
<i>set_tab</i>	hts	st	set a tab in every
			row, current columns
<i>set_window</i>	wind	wi	current window is
			lines #1-#2 cols
			#3-#4

Variable	Name	Termcap	Description
<i>tab</i>	ht	ta	tab to next 8-space
			hardware tab stop
<i>to_status_line</i>	tsl	ts	move to status line
<i>underline_char</i>	uc	uc	underline char and
			move past it
<i>up_half_line</i>	hu	hu	half a line up
<i>init_prog</i>	iprog	iP	path name of program
			for initialization

<i>key_a1</i>	ka1	K1	upper left of keypad
<i>key_a3</i>	ka3	K3	upper right of keypad
<i>key_b2</i>	kb2	K2	center of keypad
<i>key_c1</i>	kc1	K4	lower left of keypad
<i>key_c3</i>	kc3	K5	lower right of keypad
<i>prtr_non</i>	mc5p	pO	turn on printer for #1 bytes
<i>char_padding</i>	rmp	rP	like ip but when in insert mode
<i>acs_chars</i>	acsc	ac	graphics charset pairs — def=vt100
<i>plab_norm</i>	pln	pn	program label #1 to show string #2
<i>key_btab</i>	kcbt	kB	back-tab key
<i>enter_xon_mode</i>	smxon	SX	turn on xon / xoff handshaking
<i>exit_xon_mode</i>	rmxon	RX	turn off xon / xoff handshaking
<i>enter_am_mode</i>	smam	SA	turn on automatic margins
<i>exit_am_mode</i>	rmam	RA	turn off automatic margins
<i>xon_character</i>	xonc	XN	XON character
<i>xoff_character</i>	xoffc	XF	XOFF character
<i>ena_acs</i>	enacs	eA	enable alternate char set
<i>label_on</i>	smln	LO	turn on soft labels
<i>label_off</i>	rmln	LF	turn off soft labels
<i>key_beg</i>	kbeg	@1	begin key
<i>key_cancel</i>	kcan	@2	cancel key
<i>key_close</i>	kclo	@3	close key
<i>key_command</i>	kcmd	@4	command key
<i>key_copy</i>	kcpy	@5	copy key
<i>key_create</i>	kcrt	@6	create key
<i>key_end</i>	kend	@7	end key
<i>key_enter</i>	kent	@8	enter / send key
<i>key_exit</i>	kext	@9	exit key
<i>key_find</i>	kfnd	@0	find key
<i>key_help</i>	khlp	%1	help key
<i>key_mark</i>	kmrk	%2	mark key
<i>key_message</i>	kmsg	%3	message key
<i>key_move</i>	kmov	%4	move key
<i>key_next</i>	knxt	%5	next key

Variable	Name	Termcap	Description
<i>key_open</i>	kopn	%6	open key
<i>key_options</i>	kopt	%7	options key
<i>key_previous</i>	kprv	%8	previous key
<i>key_print</i>	kpri	%9	print key
<i>key_redo</i>	krdo	%0	redo key
<i>key_reference</i>	kref	&1	reference key
<i>key_refresh</i>	krfr	&2	refresh key
<i>key_replace</i>	krpl	&3	replace key
<i>key_restart</i>	krst	&4	restart key
<i>key_resume</i>	kres	&5	resume key
<i>key_save</i>	ksav	&6	save key
<i>key_suspend</i>	kspd	&7	suspend key
<i>key_undo</i>	kund	&8	undo key
<i>key_sbeg</i>	kBEG	&9	shifted key
<i>key_scancel</i>	kCAN	&0	shifted key
<i>key_scommand</i>	kCMD	*1	shifted key
<i>key_scopy</i>	kCPY	*2	shifted key
<i>key_screate</i>	kCRT	*3	shifted key
<i>key_sdc</i>	kDC	*4	shifted key
<i>key_sdl</i>	kDL	*5	shifted key
<i>key_select</i>	kslt	*6	select key
<i>key_send</i>	kEND	*7	shifted key
<i>key_seol</i>	kEOL	*8	shifted key
<i>key_sexit</i>	kEXT	*9	shifted key
<i>key_sfind</i>	kFND	*0	shifted key
<i>key_shelp</i>	kHLP	#1	shifted key
<i>key_shome</i>	kHOM	#2	shifted key
<i>key_sic</i>	kIC	#3	shifted key
<i>key_sleft</i>	kLFT	#4	shifted key
<i>key_smessage</i>	kMSG	%a	shifted key
<i>key_smove</i>	kMOV	%b	shifted key
<i>key_snext</i>	kNXT	%c	shifted key
<i>key_soptions</i>	kOPT	%d	shifted key
<i>key_sprevious</i>	kPRV	%e	shifted key
<i>key_sprint</i>	kPRT	%f	shifted key
<i>key_sredo</i>	kRDO	%g	shifted key
<i>key_sreplace</i>	kRPL	%h	shifted key
<i>key_sright</i>	kRIT	%i	shifted key
<i>key_sresume</i>	kRES	%j	shifted key
<i>key_ssav</i>	kSAV	!1	shifted key
<i>key_ssuspend</i>	kSPD	!2	shifted key
<i>key_sundo</i>	kUND	!3	shifted key
<i>req_for_input</i>	rfi	RF	send next input char (for ptys)
<i>key_f[13–24]</i>	kf[13–24]		(shift-F1 – shift-F12)

key_f[25–36] kf[25–36]
key_f[46–63] kf[46–63]

Variable	Name	Termcap	Description
<i>clr_bol</i>	el1	cb	Clear to beginning of line
<i>clear_margins</i>	mgc	MC	clear right and left soft margins
<i>set_left_margin</i>	smgl	ML	set left soft margin
<i>set_right_margin</i>	smgr	MR	set right soft margin
<i>label_format</i>	fln	Lf	label format
<i>set_clock</i>	sclk	SC	set clock, #1 hrs #2 mins #3 secs
<i>display_clock</i>	dclk	DK	display clock at (#1,#2)
<i>remove_clock</i>	rmclk	RC	remove clock
<i>create_window</i>	cwin	CW	define a window #1 from #2, #3 to #4, #5
<i>goto_window</i>	wingo	WG	go to window #1
<i>hangup</i>	hup	HU	hang-up phone
<i>dial_phone</i>	dial	DI	dial number #1
<i>quick_dial</i>	q dial	QD	dial number #1 without checking
<i>tone</i>	tone	TO	select touch tone dialing
<i>pulse</i>	pulse	PU	select pulse dialling
<i>flash_hook</i>	hook	fh	flash switch hook
<i>fixed_pause</i>	pause	PA	pause for 2-3 seconds
<i>wait_tone</i>	wait	WA	wait for dial-tone
<i>user[0–9]</i>	u[0–9]	u[0–9]	User strings #0–#9
<i>orig_pair</i>	op	op	Set default pair to its original value
<i>orig_colors</i>	oc	oc	Set all color pairs to the original ones
<i>initialize_color</i>	initc	Ic	initialize color #1 to (#2,#3,#4)
<i>initialize_pair</i>	initp	Ip	Initialize color pair #1 to fg=(#2,#3,#4), bg=(#5,#6,#7)
<i>set_color_pair</i>	scp	sp	Set current color pair to #1
<i>set_foreground</i>	setf	Sf	Set foreground color #1
<i>set_background</i>	setb	Sb	Set background color #1
<i>change_char_pitch</i>	cpi	ZA	Change number of characters per inch
<i>change_line_pitch</i>	lpi	ZB	Change number of lines per inch
<i>change_res_horz</i>	chr	ZC	Change horizontal resolution Change vertical

<i>change_res_vert</i>	cvr	ZD	resolution
<i>define_char</i>	defc	ZE	Define a character
Variable	Name	Termcap	Description
<i>enter_doublewide_mode</i>	swidm	ZF	Enter double-wide mode
<i>enter_draft_quality</i>	sdrfq	ZG	Enter draft-quality mode
<i>enter_italics_mode</i>	sitm	ZH	Enter italic mode
<i>enter_leftward_mode</i>	slm	ZI	Start leftward carriage motion
<i>enter_micro_mode</i>	smicm	ZJ	Start micro-motion mode
<i>enter_near_letter_quality</i>	snlq	ZK	Enter NLQ mode
<i>enter_normal_quality</i>	snrmq	ZL	Enter normal-quality mode
<i>enter_shadow_mode</i>	sshm	ZM	Enter shadow-print mode
<i>enter_subscript_mode</i>	ssubm	ZN	Enter subscript mode
<i>enter_superscript_mode</i>	ssupm	ZO	Enter superscript mode
<i>enter_upward_mode</i>	sum	ZP	Start upward carriage motion
<i>exit_doublewide_mode</i>	rwidm	ZQ	End double-wide mode
<i>exit_italics_mode</i>	ritm	ZR	End italic mode
<i>exit_leftward_mode</i>	rlm	ZS	End left-motion mode
<i>exit_micro_mode</i>	rmicm	ZT	End micro-motion mode
<i>exit_shadow_mode</i>	rshm	ZU	End shadow-print mode
<i>exit_subscript_mode</i>	rsubm	ZV	End subscript mode
<i>exit_superscript_mode</i>	rsupm	ZW	End superscript mode
<i>exit_upward_mode</i>	rum	ZX	End reverse character motion
<i>micro_column_address</i>	mhpa	ZY	Like column_address in micro mode
<i>micro_down</i>	mcud1	ZZ	Like cursor_down in micro mode
<i>micro_left</i>	mcub1	Za	Like cursor_left in micro mode
<i>micro_right</i>	mcuf1	Zb	Like cursor_right in micro mode
<i>micro_row_address</i>	mvpa	Zc	Like row_address in micro mode
<i>micro_up</i>	mcuu1	Zd	Like cursor_up in micro mode
<i>order_of_pins</i>	porder	Ze	Match software bits to print-head pins
<i>parm_down_micro</i>	mcud	Zf	Like parm_down_cursor in micro mode
<i>parm_left_micro</i>	mcub	Zg	Like parm_left_cursor in micro mode
<i>parm_right_micro</i>	mcuf	Zh	Like parm_right_cursor in micro mode

Variable	Name	Termcap	Description
<i>parm_up_micro</i>	mcuu	Zi	Like parm_up_cursor in micro mode
<i>select_char_set</i>	scs	Zj	Select character set
<i>set_bottom_margin</i>	smgb	Zk	Set bottom margin at current line
<i>set_bottom_margin_parm</i>	smgbp	Zl	Set bottom margin at line #1 or #2 lines from bottom
<i>set_left_margin_parm</i>	smglp	Zm	Set left (right) margin at column #1 (#2)
<i>set_right_margin_parm</i>	smgrp	Zn	Set right margin at column #1
<i>set_top_margin</i>	smgt	Zo	Set top margin at current line
<i>set_top_margin_parm</i>	smgtp	Zp	Set top (bottom) margin at row #1 (#2)
<i>start_bit_image</i>	sbim	Zq	Start printing bit image graphics
<i>start_char_set_def</i>	scsd	Zr	Start character set definition
<i>stop_bit_image</i>	rbim	Zs	Stop printing bit image graphics
<i>stop_char_set_def</i>	rcsd	Zt	End definition of character aet
<i>subscript_characters</i>	subcs	Zu	List of subscriptable characters
<i>superscript_characters</i>	supcs	Zv	List of superscriptable characters
<i>these_cause_cr</i>	docr	Zw	Printing any of these chars causes CR
<i>zero_motion</i>	zerom	Zx	No motion for subsequent character
<i>char_set_names</i>	csnm	Zy	List of character set names
<i>key_mouse</i>	kmous	Km	Mouse event has occurred
<i>mouse_info</i>	minfo	Mi	Mouse status information
<i>req_mouse_pos</i>	reqmp	RQ	Request mouse position
<i>get_mouse</i>	getm	Gm	Curses should get button events
Variable	Name	Termcap	Description
<i>set_a_foreground</i>	setaf	AF	Set ANSI foreground color
<i>set_a_background</i>	setab	AB	Set ANSI background color
<i>pkey_plab</i>	pfxl	xl	Program function key #1 to type string #2 and show string #3
<i>device_type</i>	devt	dv	Indicate language /

<i>code_set_init</i>	csin	ci	codeset support Init sequence for multiple codesets
<i>set0_des_seq</i>	s0ds	s0	Shift to code set 0 (EUC set 0, ASCII)
<i>set1_des_seq</i>	s1ds	s1	Shift to code set 1
<i>set2_des_seq</i>	s2ds	s2	Shift to code set 2
<i>set3_des_seq</i>	s3ds	s3	Shift to code set 3
<i>set_lr_margin</i>	smglr	ML	Set both left and right margins to #1, #2
<i>set_tb_margin</i>	smgtb	MT	Sets both top and bottom margins to #1, #2
<i>bit_image_repeat</i>	birep	Xy	Repeat bit image cell #1 #2 times
<i>bit_image_newline</i>	binel	Zz	Move to next row of the bit image
<i>color_names</i>	colorm	Yw	Give name for color #1
<i>set_color_band</i>	setcolor	Yz	Change to ribbon color #1
<i>set_page_length</i>	slines	YZ	Set page length to #1 lines
<i>display_pc_char</i>	dispc	S1	Display PC character
<i>enter_pc_charset_mode</i>	smpch	S2	Enter PC character display mode
<i>exit_pc_charset_mode</i>	rmpch	S3	Exit PC character display mode
<i>enter_scancode_mode</i>	smisc	S4	Enter PC scancode mode
<i>exit_scancode_mode</i>	rmsc	S5	Exit PC scancode mode
<i>pc_term_options</i>	pctrm	S6	PC terminal options
<i>scancode_escape</i>	scesc	S7	Escape for scancode emulation
<i>alt_scancode_esc</i>	scesa	S8	Alternate escape for scancode emulation
<i>bit_image_carriage_return</i>	bicr	Yv	Move to beginning of same row
<i>define_bit_image_region</i>	defbi	Yx	Define rectangular bit image region
<i>end_bit_image_region</i>	endbi	Yy	End a bit-image region

The XSI Curses standard added these. They are probably in some post-4.1 version of System V curses as well, but because XSI Curses lists them in strict alphabetical order we don't know if this is the right binary order. The ncurses termcap names for them are invented; according to the XSI Curses standard, they have no termcap names. If your compiled terminfo entries use these, they may not be binary-compatible with System V terminfo entries after SVr4.1;

beware!

Variable	Name	Termcap	Description
<i>enter_horizontal_hl_mode</i>	ehhlm	Xh	Enter horizontal highlight mode
<i>enter_left_hl_mode</i>	elhlm	Xl	Enter left highlight mode
<i>enter_low_hl_mode</i>	elohlm	Xo	Enter low highlight mode
<i>enter_right_hl_mode</i>	erhlm	Xr	Enter right highlight mode
<i>enter_top_hl_mode</i>	ethlm	Xt	Enter top highlight mode
<i>enter_vertical_hl_mode</i>	evhlm	Xv	Enter vertical highlight mode

A Sample Entry

The following entry, describing an ANSI-standard terminal, is representative of what a **terminfo** entry for a modern terminal typically looks like.

```
ansi|ansi/pc-term compatible with color,
    mc5i,
    colors#8, ncv#3, pairs#64,
    cub=\E[%p1dD, cud=\E[%p1dB, cuf=\E[%p1dC,
    cuu=\E[%p1dA, dch=\E[%p1dP, dl=\E[%p1dM,
    ech=\E[%p1dX, el1=\E[1K, hpa=\E[%p1dG, ht=\E[I,
    ich=\E[%p1d@, il=\E[%p1dL, indn=\E[%p1dS,
    kbs=^H, kcbt=\E[Z, kcub1=\E[D, kcud1=\E[B,
    kcu1=\E[C, kcuu1=\E[A, kf1=\E[M, kf10=\E[V,
    kf11=\E[W, kf12=\E[X, kf2=\E[N, kf3=\E[O, kf4=\E[P,
    kf5=\E[Q, kf6=\E[R, kf7=\E[S, kf8=\E[T, kf9=\E[U,
    kich1=\E[L, mc4=\E[4i, mc5=\E[5i, nel=\r\E[S,
    op=\E[37;40m, rep=%p1c\E[%p2%{1}%-db,
    rin=\E[%p1dT, s0ds=\E(B, s1ds=\E)B, s2ds=\E*B,
    s3ds=\E+B, setab=\E[4%p1dm, setaf=\E[3%p1dm,
    setb=\E[4%?p1%{1}%=%t4e%p1%{3}%=%t6e%p1%{4}%=%t1e%p1%{6}%=%t3e%p1d%;m,
    setf=\E[3%?p1%{1}%=%t4e%p1%{3}%=%t6e%p1%{4}%=%t1e%p1%{6}%=%t3e%p1d%;m,
    sgr=\E[0;10%?p1t;7%;%?p2t;4%;%?p3t;7%;%?p4t;5%;%?p6t;1%;%?p7t;8%;%?p8t;9%;%?p9t;10%;%?p10t;11%;%?p11t;12%;%?p12t;13%;%?p13t;14%;%?p14t;15%;%?p15t;16%;%?p16t;17%;%?p17t;18%;%?p18t;19%;%?p19t;20%;%?p20t;21%;%?p21t;22%;%?p22t;23%;%?p23t;24%;%?p24t;25%;%?p25t;26%;%?p26t;27%;%?p27t;28%;%?p28t;29%;%?p29t;30%;%?p30t;31%;%?p31t;32%;%?p32t;33%;%?p33t;34%;%?p34t;35%;%?p35t;36%;%?p36t;37%;%?p37t;38%;%?p38t;39%;%?p39t;40%;%?p40t;41%;%?p41t;42%;%?p42t;43%;%?p43t;44%;%?p44t;45%;%?p45t;46%;%?p46t;47%;%?p47t;48%;%?p48t;49%;%?p49t;50%;%?p50t;51%;%?p51t;52%;%?p52t;53%;%?p53t;54%;%?p54t;55%;%?p55t;56%;%?p56t;57%;%?p57t;58%;%?p58t;59%;%?p59t;60%;%?p60t;61%;%?p61t;62%;%?p62t;63%;%?p63t;64%;%?p64t;65%;%?p65t;66%;%?p66t;67%;%?p67t;68%;%?p68t;69%;%?p69t;70%;%?p70t;71%;%?p71t;72%;%?p72t;73%;%?p73t;74%;%?p74t;75%;%?p75t;76%;%?p76t;77%;%?p77t;78%;%?p78t;79%;%?p79t;80%;%?p80t;81%;%?p81t;82%;%?p82t;83%;%?p83t;84%;%?p84t;85%;%?p85t;86%;%?p86t;87%;%?p87t;88%;%?p88t;89%;%?p89t;90%;%?p90t;91%;%?p91t;92%;%?p92t;93%;%?p93t;94%;%?p94t;95%;%?p95t;96%;%?p96t;97%;%?p97t;98%;%?p98t;99%;%?p99t;100%;%?p100t;101%;%?p101t;102%;%?p102t;103%;%?p103t;104%;%?p104t;105%;%?p105t;106%;%?p106t;107%;%?p107t;108%;%?p108t;109%;%?p109t;110%;%?p110t;111%;%?p111t;112%;%?p112t;113%;%?p113t;114%;%?p114t;115%;%?p115t;116%;%?p116t;117%;%?p117t;118%;%?p118t;119%;%?p119t;120%;%?p120t;121%;%?p121t;122%;%?p122t;123%;%?p123t;124%;%?p124t;125%;%?p125t;126%;%?p126t;127%;%?p127t;128%;%?p128t;129%;%?p129t;130%;%?p130t;131%;%?p131t;132%;%?p132t;133%;%?p133t;134%;%?p134t;135%;%?p135t;136%;%?p136t;137%;%?p137t;138%;%?p138t;139%;%?p139t;140%;%?p140t;141%;%?p141t;142%;%?p142t;143%;%?p143t;144%;%?p144t;145%;%?p145t;146%;%?p146t;147%;%?p147t;148%;%?p148t;149%;%?p149t;150%;%?p150t;151%;%?p151t;152%;%?p152t;153%;%?p153t;154%;%?p154t;155%;%?p155t;156%;%?p156t;157%;%?p157t;158%;%?p158t;159%;%?p159t;160%;%?p160t;161%;%?p161t;162%;%?p162t;163%;%?p163t;164%;%?p164t;165%;%?p165t;166%;%?p166t;167%;%?p167t;168%;%?p168t;169%;%?p169t;170%;%?p170t;171%;%?p171t;172%;%?p172t;173%;%?p173t;174%;%?p174t;175%;%?p175t;176%;%?p176t;177%;%?p177t;178%;%?p178t;179%;%?p179t;180%;%?p180t;181%;%?p181t;182%;%?p182t;183%;%?p183t;184%;%?p184t;185%;%?p185t;186%;%?p186t;187%;%?p187t;188%;%?p188t;189%;%?p189t;190%;%?p190t;191%;%?p191t;192%;%?p192t;193%;%?p193t;194%;%?p194t;195%;%?p195t;196%;%?p196t;197%;%?p197t;198%;%?p198t;199%;%?p199t;200%;%?p200t;201%;%?p201t;202%;%?p202t;203%;%?p203t;204%;%?p204t;205%;%?p205t;206%;%?p206t;207%;%?p207t;208%;%?p208t;209%;%?p209t;210%;%?p210t;211%;%?p211t;212%;%?p212t;213%;%?p213t;214%;%?p214t;215%;%?p215t;216%;%?p216t;217%;%?p217t;218%;%?p218t;219%;%?p219t;220%;%?p220t;221%;%?p221t;222%;%?p222t;223%;%?p223t;224%;%?p224t;225%;%?p225t;226%;%?p226t;227%;%?p227t;228%;%?p228t;229%;%?p229t;230%;%?p230t;231%;%?p231t;232%;%?p232t;233%;%?p233t;234%;%?p234t;235%;%?p235t;236%;%?p236t;237%;%?p237t;238%;%?p238t;239%;%?p239t;240%;%?p240t;241%;%?p241t;242%;%?p242t;243%;%?p243t;244%;%?p244t;245%;%?p245t;246%;%?p246t;247%;%?p247t;248%;%?p248t;249%;%?p249t;250%;%?p250t;251%;%?p251t;252%;%?p252t;253%;%?p253t;254%;%?p254t;255%;%?p255t;256%;%?p256t;257%;%?p257t;258%;%?p258t;259%;%?p259t;260%;%?p260t;261%;%?p261t;262%;%?p262t;263%;%?p263t;264%;%?p264t;265%;%?p265t;266%;%?p266t;267%;%?p267t;268%;%?p268t;269%;%?p269t;270%;%?p270t;271%;%?p271t;272%;%?p272t;273%;%?p273t;274%;%?p274t;275%;%?p275t;276%;%?p276t;277%;%?p277t;278%;%?p278t;279%;%?p279t;280%;%?p280t;281%;%?p281t;282%;%?p282t;283%;%?p283t;284%;%?p284t;285%;%?p285t;286%;%?p286t;287%;%?p287t;288%;%?p288t;289%;%?p289t;290%;%?p290t;291%;%?p291t;292%;%?p292t;293%;%?p293t;294%;%?p294t;295%;%?p295t;296%;%?p296t;297%;%?p297t;298%;%?p298t;299%;%?p299t;300%;%?p300t;301%;%?p301t;302%;%?p302t;303%;%?p303t;304%;%?p304t;305%;%?p305t;306%;%?p306t;307%;%?p307t;308%;%?p308t;309%;%?p309t;310%;%?p310t;311%;%?p311t;312%;%?p312t;313%;%?p313t;314%;%?p314t;315%;%?p315t;316%;%?p316t;317%;%?p317t;318%;%?p318t;319%;%?p319t;320%;%?p320t;321%;%?p321t;322%;%?p322t;323%;%?p323t;324%;%?p324t;325%;%?p325t;326%;%?p326t;327%;%?p327t;328%;%?p328t;329%;%?p329t;330%;%?p330t;331%;%?p331t;332%;%?p332t;333%;%?p333t;334%;%?p334t;335%;%?p335t;336%;%?p336t;337%;%?p337t;338%;%?p338t;339%;%?p339t;340%;%?p340t;341%;%?p341t;342%;%?p342t;343%;%?p343t;344%;%?p344t;345%;%?p345t;346%;%?p346t;347%;%?p347t;348%;%?p348t;349%;%?p349t;350%;%?p350t;351%;%?p351t;352%;%?p352t;353%;%?p353t;354%;%?p354t;355%;%?p355t;356%;%?p356t;357%;%?p357t;358%;%?p358t;359%;%?p359t;360%;%?p360t;361%;%?p361t;362%;%?p362t;363%;%?p363t;364%;%?p364t;365%;%?p365t;366%;%?p366t;367%;%?p367t;368%;%?p368t;369%;%?p369t;370%;%?p370t;371%;%?p371t;372%;%?p372t;373%;%?p373t;374%;%?p374t;375%;%?p375t;376%;%?p376t;377%;%?p377t;378%;%?p378t;379%;%?p379t;380%;%?p380t;381%;%?p381t;382%;%?p382t;383%;%?p383t;384%;%?p384t;385%;%?p385t;386%;%?p386t;387%;%?p387t;388%;%?p388t;389%;%?p389t;390%;%?p390t;391%;%?p391t;392%;%?p392t;393%;%?p393t;394%;%?p394t;395%;%?p395t;396%;%?p396t;397%;%?p397t;398%;%?p398t;399%;%?p399t;400%;%?p400t;401%;%?p401t;402%;%?p402t;403%;%?p403t;404%;%?p404t;405%;%?p405t;406%;%?p406t;407%;%?p407t;408%;%?p408t;409%;%?p409t;410%;%?p410t;411%;%?p411t;412%;%?p412t;413%;%?p413t;414%;%?p414t;415%;%?p415t;416%;%?p416t;417%;%?p417t;418%;%?p418t;419%;%?p419t;420%;%?p420t;421%;%?p421t;422%;%?p422t;423%;%?p423t;424%;%?p424t;425%;%?p425t;426%;%?p426t;427%;%?p427t;428%;%?p428t;429%;%?p429t;430%;%?p430t;431%;%?p431t;432%;%?p432t;433%;%?p433t;434%;%?p434t;435%;%?p435t;436%;%?p436t;437%;%?p437t;438%;%?p438t;439%;%?p439t;440%;%?p440t;441%;%?p441t;442%;%?p442t;443%;%?p443t;444%;%?p444t;445%;%?p445t;446%;%?p446t;447%;%?p447t;448%;%?p448t;449%;%?p449t;450%;%?p450t;451%;%?p451t;452%;%?p452t;453%;%?p453t;454%;%?p454t;455%;%?p455t;456%;%?p456t;457%;%?p457t;458%;%?p458t;459%;%?p459t;460%;%?p460t;461%;%?p461t;462%;%?p462t;463%;%?p463t;464%;%?p464t;465%;%?p465t;466%;%?p466t;467%;%?p467t;468%;%?p468t;469%;%?p469t;470%;%?p470t;471%;%?p471t;472%;%?p472t;473%;%?p473t;474%;%?p474t;475%;%?p475t;476%;%?p476t;477%;%?p477t;478%;%?p478t;479%;%?p479t;480%;%?p480t;481%;%?p481t;482%;%?p482t;483%;%?p483t;484%;%?p484t;485%;%?p485t;486%;%?p486t;487%;%?p487t;488%;%?p488t;489%;%?p489t;490%;%?p490t;491%;%?p491t;492%;%?p492t;493%;%?p493t;494%;%?p494t;495%;%?p495t;496%;%?p496t;497%;%?p497t;498%;%?p498t;499%;%?p499t;500%;%?p500t;501%;%?p501t;502%;%?p502t;503%;%?p503t;504%;%?p504t;505%;%?p505t;506%;%?p506t;507%;%?p507t;508%;%?p508t;509%;%?p509t;510%;%?p510t;511%;%?p511t;512%;%?p512t;513%;%?p513t;514%;%?p514t;515%;%?p515t;516%;%?p516t;517%;%?p517t;518%;%?p518t;519%;%?p519t;520%;%?p520t;521%;%?p521t;522%;%?p522t;523%;%?p523t;524%;%?p524t;525%;%?p525t;526%;%?p526t;527%;%?p527t;528%;%?p528t;529%;%?p529t;530%;%?p530t;531%;%?p531t;532%;%?p532t;533%;%?p533t;534%;%?p534t;535%;%?p535t;536%;%?p536t;537%;%?p537t;538%;%?p538t;539%;%?p539t;540%;%?p540t;541%;%?p541t;542%;%?p542t;543%;%?p543t;544%;%?p544t;545%;%?p545t;546%;%?p546t;547%;%?p547t;548%;%?p548t;549%;%?p549t;550%;%?p550t;551%;%?p551t;552%;%?p552t;553%;%?p553t;554%;%?p554t;555%;%?p555t;556%;%?p556t;557%;%?p557t;558%;%?p558t;559%;%?p559t;560%;%?p560t;561%;%?p561t;562%;%?p562t;563%;%?p563t;564%;%?p564t;565%;%?p565t;566%;%?p566t;567%;%?p567t;568%;%?p568t;569%;%?p569t;570%;%?p570t;571%;%?p571t;572%;%?p572t;573%;%?p573t;574%;%?p574t;575%;%?p575t;576%;%?p576t;577%;%?p577t;578%;%?p578t;579%;%?p579t;580%;%?p580t;581%;%?p581t;582%;%?p582t;583%;%?p583t;584%;%?p584t;585%;%?p585t;586%;%?p586t;587%;%?p587t;588%;%?p588t;589%;%?p589t;590%;%?p590t;591%;%?p591t;592%;%?p592t;593%;%?p593t;594%;%?p594t;595%;%?p595t;596%;%?p596t;597%;%?p597t;598%;%?p598t;599%;%?p599t;600%;%?p600t;601%;%?p601t;602%;%?p602t;603%;%?p603t;604%;%?p604t;605%;%?p605t;606%;%?p606t;607%;%?p607t;608%;%?p608t;609%;%?p609t;610%;%?p610t;611%;%?p611t;612%;%?p612t;613%;%?p613t;614%;%?p614t;615%;%?p615t;616%;%?p616t;617%;%?p617t;618%;%?p618t;619%;%?p619t;620%;%?p620t;621%;%?p621t;622%;%?p622t;623%;%?p623t;624%;%?p624t;625%;%?p625t;626%;%?p626t;627%;%?p627t;628%;%?p628t;629%;%?p629t;630%;%?p630t;631%;%?p631t;632%;%?p632t;633%;%?p633t;634%;%?p634t;635%;%?p635t;636%;%?p636t;637%;%?p637t;638%;%?p638t;639%;%?p639t;640%;%?p640t;641%;%?p641t;642%;%?p642t;643%;%?p643t;644%;%?p644t;645%;%?p645t;646%;%?p646t;647%;%?p647t;648%;%?p648t;649%;%?p649t;650%;%?p650t;651%;%?p651t;652%;%?p652t;653%;%?p653t;654%;%?p654t;655%;%?p655t;656%;%?p656t;657%;%?p657t;658%;%?p658t;659%;%?p659t;660%;%?p660t;661%;%?p661t;662%;%?p662t;663%;%?p663t;664%;%?p664t;665%;%?p665t;666%;%?p666t;667%;%?p667t;668%;%?p668t;669%;%?p669t;670%;%?p670t;671%;%?p671t;672%;%?p672t;673%;%?p673t;674%;%?p674t;675%;%?p675t;676%;%?p676t;677%;%?p677t;678%;%?p678t;679%;%?p679t;680%;%?p680t;681%;%?p681t;682%;%?p682t;683%;%?p683t;684%;%?p684t;685%;%?p685t;686%;%?p686t;687%;%?p687t;688%;%?p688t;689%;%?p689t;690%;%?p690t;691%;%?p691t;692%;%?p692t;693%;%?p693t;694%;%?p694t;695%;%?p695t;696%;%?p696t;697%;%?p697t;698%;%?p698t;699%;%?p699t;700%;%?p700t;701%;%?p701t;702%;%?p702t;703%;%?p703t;704%;%?p704t;705%;%?p705t;706%;%?p706t;707%;%?p707t;708%;%?p708t;709%;%?p709t;710%;%?p710t;711%;%?p711t;712%;%?p712t;713%;%?p713t;714%;%?p714t;715%;%?p715t;716%;%?p716t;717%;%?p717t;718%;%?p718t;719%;%?p719t;720%;%?p720t;721%;%?p721t;722%;%?p722t;723%;%?p723t;724%;%?p724t;725%;%?p725t;726%;%?p726t;727%;%?p727t;728%;%?p728t;729%;%?p729t;730%;%?p730t;731%;%?p731t;732%;%?p732t;733%;%?p733t;734%;%?p734t;735%;%?p735t;736%;%?p736t;737%;%?p737t;738%;%?p738t;739%;%?p739t;740%;%?p740t;741%;%?p741t;742%;%?p742t;743%;%?p743t;744%;%?p744t;745%;%?p745t;746%;%?p746t;747%;%?p747t;748%;%?p748t;749%;%?p749t;750%;%?p750t;751%;%?p751t;752%;%?p752t;753%;%?p753t;754%;%?p754t;755%;%?p755t;756%;%?p756t;757%;%?p757t;758%;%?p758t;759%;%?p759t;760%;%?p760t;761%;%?p761t;762%;%?p762t;763%;%?p763t;764%;%?p764t;765%;%?p765t;766%;%?p766t;767%;%?p767t;768%;%?p768t;769%;%?p769t;770%;%?p770t;771%;%?p771t;772%;%?p772t;773%;%?p773t;774%;%?p774t;775%;%?p775t;776%;%?p776t;777%;%?p777t;778%;%?p778t;779%;%?p779t;780%;%?p780t;781%;%?p781t;782%;%?p782t;783%;%?p783t;784%;%?p784t;785%;%?p785t;786%;%?p786t;787%;%?p787t;788%;%?p788t;789%;%?p789t;790%;%?p790t;791%;%?p791t;792%;%?p792t;793%;%?p793t;794%;%?p794t;795%;%?p795t;796%;%?p796t;797%;%?p797t;798%;%?p798t;799%;%?p799t;800%;%?p800t;801%;%?p801t;802%;%?p802t;803%;%?p803t;804%;%?p804t;805%;%?p805t;806%;%?p806t;807%;%?p807t;808%;%?p808t;809%;%?p809t;810%;%?p810t;811%;%?p811t;812%;%?p812t;813%;%?p813t;814%;%?p814t;815%;%?p815t;816%;%?p816t;817%;%?p817t;818%;%?p818t;819%;%?p819t;820%;%?p820t;821%;%?p821t;822%;%?p822t;823%;%?p823t;824%;%?p824t;825%;%?p825t;826%;%?p826t;827%;%?p827t;828%;%?p828t;829%;%?p829t;830%;%?p830t;831%;%?p831t;832%;%?p832t;833%;%?p833t;834%;%?p834t;835%;%?p835t;836%;%?p836t;837%;%?p837t;838%;%?p838t;839%;%?p839t;840%;%?p840t;841%;%?p841t;842%;%?p842t;843%;%?p843t;844%;%?p844t;845%;%?p845t;846%;%?p846t;847%;%?p847t;848%;%?p848t;849%;%?p849t;850%;%?p850t;851%;%?p851t;852%;%?p852t;853%;%?p853t;854%;%?p854t;855%;%?p855t;856%;%?p856t;857%;%?p857t;858%;%?p858t;859%;%?p859t;860%;%?p860t;861%;%?p861t;862%;%?p862t;863%;%?p863t;864%;%?p864t;865%;%?p865t;866%;%?p866t;867%;%?p867t;868%;%?p868t;869%;%?p869t;870%;%?p870t;871%;%?p871t;872%;%?p872t;873%;%?p873t;874%;%?p874t;875%;%?p875t;876%;%?p876t;877%;%?p877t;878%;%?p878t;879%;%?p879t;880%;%?p880t;881%;%?p881t;882%;%?p882t;883%;%?p883t;884%;%?p884t;885%;%?p885t;886%;%?p886t;887%;%?p887t;888%;%?p888t;889%;%?p889t;890%;%?p890t;891%;%?p891t;892%;%?p892t;893%;%?p893t;894%;%?p894t;895%;%?p895t;896%;%?p896t;897%;%?p897t;898%;%?p898t;899%;%?p899t;900%;%?p900t;901%;%?p901t;902%;%?p902t;903%;%?p903t;904%;%?p904t;905%;%?p905t;906%;%?p906t;907%;%?p907t;908%;%?p908t;909%;%?p909t;910%;%?p910t;911%;%?p911t;912%;%?p912t;913%;%?p913t;914%;%?p914t;915%;%?p915t;916%;%?p916t;917%;%?p917t;918%;%?p918t;919%;%?p919t;920%;%?p920t;921%;%?p921t;922%;%?p922t;923%;%?p923t;924%;%?p924t;925%;%?p925t;926%;%?p926t;927%;%?p927t;928%;%?p928t;929%;%?p929t;930%;%?p930t;931%;%?p931t;932%;%?p932t;933%;%?p933t;934%;%?p934t;935%;%?p935t;936%;%?p936t;937%;%?p937t;938%;%?p938t;939%;%?p939t;940%;%?p940t;941%;%?p941t;942%;%?p942t;943%;%?p943t;944%;%?p944t;945%;%?p945t;946%;%?p946t;947%;%?p947t;948%;%?p948t;949%;%?p949t;950%;%?p950t;951%;%?p951t;952%;%?p952t;953%;%?p953t;954%;%?p954t;955%;%?p955t;956%;%?p956t;957%;%?p957t;958%;%?p958t;959%;%?p959t;960%;%?p
```

Types of Capabilities

All capabilities have names. For instance, the fact that ANSI-standard terminals have *automatic margins* (i.e., an automatic return and line-feed when the end of a line is reached) is indicated by the capability **am**. Hence the description of **ansi** includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **cols**, which indicates the number of columns the terminal has, gives the value '80' for **ansi**.

Finally, string valued capabilities, such as **el** (clear to end of line sequence) are given by the two-character code, an '=', and then a string ending at the next following ','.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. Both **\E** and **\e** map to an ESCAPE character, **\x** maps to a control-x for any appropriate x, and the sequences **\n**, **\l**, **\r**, **\t**, **\b**, **\f**, and **\s** give a newline, line-feed, return, tab, backspace, form-feed, and space. Other escapes include **\^** for **^**, **** for ****, **\,** for comma, **\:** for **:**, and **\0** for null. (**\0** will produce **\200**, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a ****.

A delay in milliseconds may appear anywhere in a string capability, enclosed in **\$<..>** brackets, as in **el=\EK\$<5>**, and padding characters are supplied by **tputs()** to provide this delay. The delay must be a number with at most one decimal place of precision; it may be followed by suffixes '*' or '/' or both. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of *lines* affected.) Normally, padding is advisory if the device has the **xon** capability; it is used for cost computation but does not trigger delays. A '/' suffix indicates that the padding is mandatory and forces a delay of the given number of milliseconds even on devices for which **xon** is present to indicate flow control.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above.

Fetching Compiled Descriptions

To easily test a new terminal description you can set the environment variable **TERMINFO** to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in **terminfo**.

If **TERMINFO** is not set, the ncurses version of the terminfo reader code will instead look in the directory **\$HOME/.terminfo** for a compiled description. (This extension is not supported under stock System V terminfo/curses.)

If these local lookups fail, the last place tried will be the system terminfo directory, **terminfo**.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *terminfo* and to build up a description gradually, using partial descriptions with *vi* or some other screen-oriented program to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *terminfo* file to describe it or bugs in the screen-handling code of the test program.

To get the padding for insert line right (if the terminal manufacturer did not document it) a severe test is to edit `/etc/passwd` at 9600 baud, delete 16 or so lines from the middle of the screen, then hit the 'u' key several times quickly. If the terminal messes up, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as TEKTRONIX 4010 series, as well as hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage return, control M.) If there is a code to produce an audible signal (bell, beep, etc) give this as **bel**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over, for example, you would not normally use '**cuf1**=\ ' because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are

undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the *terminfo* file usually assumes that this is on; i.e., **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hard-copy and “glass-tty” terminals. Thus the model 33 teletype is described as

```
33|tty33|tty|model 33 teletype,
bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM-3 is described as

```
adm3|3|lsi adm3,
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,
ind=^J, lines#24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with *printf()*-like escapes **%x** in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special **%** codes to manipulate it. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary.

The **%** encodings have the following meanings:

%%	outputs ‘%’
%d	print pop() as in printf

%2d	print pop() like %2d
%3d	print pop() like %3d
%02d	
%03d	as in printf
%x	print pop() as in printf
%2x	print pop() like %2x
%3x	print pop() like %3x
%02x	
%03x	as in printf
%c	print pop() gives %c
%s	print pop() gives %s
%p[1-9]	push ith parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'c'	char constant c
%{nn}	integer constant nn
%+ %- %* %/ %m	arithmetic (%m is mod): push(pop() op pop())
%& % %^	bit operations: push(pop() op pop())
%= %> %<	logical operations: push(pop() op pop())
%A, %O	logical and & or operations (for conditionals)
%! %	unary operations push(op pop())
%i	add 1 to first two parms (for ANSI terminals)
%?	expr %t thenpart %e elsepart %;
	if-then-else, %e elsepart is optional.
	else-if's are possible ala Algol 68:
	%%? c ₁ %t b ₁ %e c ₂ %t b ₂ %e c ₃ %t b ₃ %e c ₄ %t b ₄ %e %;

c_i are conditions, b_i are bodies.

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use "%gx%{5}%-".

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cup** capability is "cup=6\E&%p2%2dc%p1%2dY".

The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, "cup=^T%p1%c%p2%c". Terminals which use "%c" need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit \n ^D and \r, as the system may change or discard them. (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus "cup=\E=%p1%' '%+%c%p2%' '%+%c". After sending '\E=', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second

parameter. More complex arithmetic is possible using the stack.

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two parameter sequence (as with the hp2645) and can be used in preference to **cup**. If there are parameterized local motions (e.g., move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the TEKTRONIX 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the \EH sequence on HP terminals cannot be used for **home**.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **Ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Insert/delete line and vertical motions

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**. If the terminal has a settable scrolling region (like the vt100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command.

It is possible to get the effect of insert or delete line using **csr** on a properly chosen region; the **sc** and **rc** (save and restore cursor) commands may be useful for ensuring that your synthesized insert/delete string does not move the cursor. (Note that the ncurses(3x) library does this synthesis automatically, so you need not compose insert/delete strings for an entry with **csr**).

Yet another way to construct insert and delete might be to use a combination of index with the memory-lock feature found on some terminals (like the HP-700/90 series, which however also has insert/delete).

Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert / delete line, and is often faster even on terminals with those features.

The boolean **non_dest_scroll_region** should be set if each scrolling window is effectively a viewport on a screen-sized canvas. To test for this capability, create a scrolling region in the middle of the screen, write something to the bottom line, move the cursor to the top of the region, and do **ri** followed by **dl1** or **ind**. If the data scrolled off the bottom of the region by the **ri** re-appears, then scrolling is non-destructive. System V and XSI Curses expect that **ind**, **ri**, **indn**, and **rin** will simulate destructive scrolling; their documentation cautions you not to define **csr** unless this is true. This **curses** implementation is more liberal and will do explicit erases after scrolling if **ndstr** is defined.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert / delete character which can be described using *terminfo*. The most common insert / delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type “abc def” using local cursor motions (not spaces) between the “abc” and the “def”. Then position the cursor before the “abc” and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the “abc” shifts over to the “def” which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for “insert null”. While these are two logically separate attributes (one line vs. multi-line insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

Terminfo can describe both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals which send a sequence to open a

screen position should give it here.

If your terminal has both, insert mode is usually preferable to **ich1**. Technically, you should not give both unless the terminal actually requires both to be used in combination. Accordingly, some non-curses applications get confused if both are present; the symptom is doubled characters in an update using insert. This requirement is now rare; most **ich** sequences do not require previous **smir**, and most **smir** insert modes do not require **ich1** before each character. Therefore, the new **curses** actually assumes this is the case and uses either **rmir/smirt** or **ich/ich1** as appropriate (but not both). If you have to write an entry to be used under new curses for a terminal old enough to need both, include the **rmir/smirt** sequences in **ich1**.

If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will repeat the effects of **ich1** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode*, representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking) **bold** (bold or extra bright) **dim** (dim or half-bright) **invis** (blanking or invisible text) **prot** (protected) **rev** (reverse video) **sgr0** (turn off *all* attribute modes)

smacs (enter alternate character set mode) and **rmacs** (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist.

Terminals with the “magic cookie” glitch (**xmc**) deposit special “cookies” when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **flash**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of both of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the TEKTRONIX 4025, where **smcup** sets the command character to be the one used by terminfo.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad Handling

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1**, **kcuf1**, **kcuu1**, **kcud1**, and **khome** respectively. If there are function keys such as f0, f1, ..., f10, the codes they send can be given as **kf0**, **kf1**, ..., **kf10**. If these keys have labels other than the default f0 through f10, the labels can be given as **lf0**, **lf1**, ..., **lf10**. The codes

transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knnp** (next page), **kpp** (previous page), **kind** (scroll forward / down), **kri** (scroll backward / up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A “back-tab” command which moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by the *tset* command to determine whether to set the mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in non-volatile memory, the terminfo description can assume that they are properly set.

Other capabilities include **is1**, **is2**, and **is3**, initialization strings for the terminal, **iprog**, the path name of a program to be run to initialize the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the terminfo description. They are normally sent to the terminal, by the *tset* program, each time the user logs in. They will be printed in the following order: **is1**; **is2**; setting tabs using **tbc** and **hts**; **if**; running the program **iprog**; and finally **is3**. Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is2** and **if**. These strings are output by the *reset* program, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs2** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the vt100 into 80-column mode would normally be part of **is2**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80 column mode.

If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is2** or **if**.

Delays and Padding

Many older and slower terminals don't support either XON / XOFF or DTR handshaking, including hardcopy terminals and some very archaic CRTs (including, for example, DEC VT100s). These may require padding characters after certain cursor motions and screen changes.

If the terminal uses xon / xoff handshaking for flow control (that is, it automatically emits ^S back to the host when its input buffers are close to full), set **xon**. This capability suppresses the emission of padding. You can also set it for memory-mapped console devices effectively that don't have a speed limit. Padding information should still be included so that routines can make better decisions about relative costs, but actual pad characters will not be transmitted.

If **pb** (padding baud rate) is given, padding is suppressed at baud rates below the value of **pb**. If the entry has no padding baud rate, then whether padding is emitted or not is completely controlled by **xon**.

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used.

Status Lines

Some terminals have an extra 'status line' which is not normally used by software (and thus not counted in the terminal's **lines** capability).

The simplest case is a status line which is cursor-addressable but not part of the main scrolling region on the screen; the Heathkit H19 has a status line of this kind, as would a 24-line VT100 with a 23-line scrolling region set up on initialization. This situation is indicated by the **hs** capability.

Some terminals with status lines need special sequences to access the status line. These may be expressed as a string with single parameter **tsl** which takes the cursor to a given zero-origin column on the status line. The capability **fsl** must return to the main-screen cursor positions before the last **tsl**. You may need to embed the string values of **sc** (save cursor) and **rc** (restore cursor) in **tsl** and **fsl** to accomplish this.

The status line is normally assumed to be the same width as the width of the terminal. If this is untrue, you can specify it with the numeric capability **wsl**.

A command to erase or blank the status line may be specified as **dsl**.

The boolean capability **eslok** specifies that escape sequences, tabs, etc. work ordinarily in the status line.

The ncurses implementation does not yet use any of these capabilities. They are documented here in case they ever become important.

Line Graphics

Many terminals have alternate character sets useful for forms-drawing. Terminfo and **curses** build in support for the drawing characters supported by the VT100, with some characters from the AT&T 4410v1 added. This alternate character set may be specified by the **acsc** capability.

Glyph Name	ACS Name	Ascii Default	VT100 Name
upper left corner	ACS_ULCORNER	+	l
lower left corner	ACS_LLCORNER	+	m
upper right corner	ACS_URCORNER	+	k
lower right corner	ACS_LRCORNER	+	j
tee pointing right	ACS_LTEE	+	t
tee pointing left	ACS_RTEE	+	u
tee pointing up	ACS_BTEE	+	v
tee pointing down	ACS_TTEE	+	w
horizontal line	ACS_HLINE	-	q
vertical line	ACS_VLINE		x
large plus or crossover	ACS_PLUS	+	n
scan line 1	ACS_S1		o
scan line 9	ACS_S9		s
diamond	ACS_DIAMOND	+	‘
checker board (stipple)	ACS_CKBOARD	:	a
degree symbol	ACS_DEGREE	\	f
plus / minus	ACS_PLMINUS	#	g
bullet	ACS_BULLET	o	
arrow pointing left	ACS_LARROW	<	,
arrow pointing right	ACS_RARROW	>	+
arrow pointing down	ACS_DARROW	v	.
arrow pointing up	ACS_UARROW	^	-
board of squares	ACS_BOARD	#	h
lantern symbol	ACS_LANTERN	#	I
solid square block	ACS_BLOCK	#	0

The best way to define a new device's graphics set is to add a column to a copy of this table for your terminal, giving the character which (when emitted between **smacs** / **rmacs** switches) will be rendered as the corresponding graphic. Then read off the VT100/your terminal character pairs right to left in sequence; these become the ACSC string.

Color Handling

Most color terminals are either 'Tektronix-like' or 'HP-like'. Tektronix-like terminals have a predefined set of N colors (where N usually 8), and can set character-cell foreground and background characters independently, mixing them into N * N color-pairs. On HP-like terminals, the user must set each color pair up separately (foreground and background are not independently settable). Up to M color-pairs may be set up from 2*M different colors. ANSI-compatible terminals are Tektronix-like.

Some basic color capabilities are independent of the color method. The numeric capabilities **colors** and **pairs** specify the maximum numbers of colors and color-pairs that can be displayed simultaneously. The **op** (original pair)

string resets foreground and background colors to their default values for the terminal. The **oc** string resets all colors or color-pairs to their default values for the terminal. Some terminals (including many PC terminal emulators) erase screen areas with the current background color rather than the power-up default background; these should have the boolean capability **bce**.

To change the current foreground or background color on a Tektronix-type terminal, use **setaf** (set ANSI foreground) and **setab** (set ANSI background). These take one parameter, the color number. On an HP-like terminal, use **scp** with a color-pair number parameter to set which color pair is current.

On a Tektronix-like terminal, the capability **ccc** may be present to indicate that colors can be modified. If so, the **initc** capability will take a color number (0 to **colors** - 1) and three more parameters which describe the color. These three parameters default to being interpreted as RGB (Red, Green, Blue) values. If the boolean capability **hls** is present, they are instead as HLS (Hue, Lightness, Saturation) indices. The ranges are terminal-dependent.

On an HP-like terminal, **initp** may give a capability for changing a color-pair value. It will take seven parameters; a color-pair number (0 to **max_pairs** - 1), and two triples describing first background and then foreground colors. These parameters must be (Red, Green, Blue) or (Hue, Lightness, Saturation) depending on **hls**.

On some color terminals, colors collide with highlights. You can register these collisions with the **ncv** capability. This is a bit-mask of attributes not to be used when colors are enabled. The correspondence with the attributes understood by **curses** is as follows:

Attribute	Bit	Decimal
A_STANDOUT	0	1
A_UNDERLINE	1	2
A_REVERSE	2	4
A_BLINK	3	8
A_DIM	4	16
A_BOLD	5	32
A_INVIS	6	64
A_PROTECT	7	128
A_ALTCHARSET	8	256

For example, on many IBM PC consoles, the underline attribute collides with the foreground color blue and is not available in color mode. These should have an **ncv** capability of 2.

Miscellaneous

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hard-copy terminals. If a hard-copy terminal can eject to the next page (form feed), give this as **ff** (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the

character to be repeated and the second is the number of times to repeat it. Thus, `tparm(repeat_char, 'x', 10)` is the same as `'xxxxxxxxxx'`.

If the terminal has a settable command character, such as the TEKTRONIX 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some systems: The environment is to be searched for a **CC** variable, and if found, all occurrences of the prototype character are replaced with the character in the environment variable.

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.)

If the terminal has a “meta key” which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this “meta mode” on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the virtual terminal protocol, the terminal number can be given as **vt**.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **mc5p** takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local; and **pfx** causes the string to be transmitted to the computer.

Glitches and Braindamage

Hazeltine terminals, which do not allow “ characters to be displayed should indicate **hz**.

Terminals which ignore a line-feed immediately after an **am** wrap, such as the Concept and vt100, should indicate **xenl**.

If **el** is required to get rid of standout (instead of merely writing normal text on top of it), **xhp** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). Note: the variable indicating this is now ‘dest_tabs_magic_sms0’; in older versions, it was teleray_glitch. This glitch is also taken to mean that it is not possible to position the cursor on top of a “magic cookie”, that to erase standout mode it is instead necessary to use delete and insert line. The ncurses implementation ignores this glitch.

The Beehive Superbee, which is unable to correctly transmit the escape or control C characters, has **xsfb**, indicating that the f1 key is used for escape and f2 for control C. (Only certain Superbees have this problem, depending on the ROM.) Note that in older terminfo versions, this capability was called ‘beehive_glitch’; it is now ‘no_esc_ctl_c’.

Other specific terminal problems may be corrected by adding more capabilities of the form **xx**.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be cancelled by placing **xx@** to the left of the capability definition, where **xx** is the capability. For example, the entry

```
2621-nl, smkx@, rmkx@, use=2621,
```

defines a 2621-nl that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

Pitfalls of Long Entries

Long terminfo entries are unlikely to be a problem; to date, no entry has even approached terminfo’s 4K string-table maximum. Unfortunately, the termcap translations are much more strictly limited (to 1K), thus termcap translations of long terminfo entries can cause problems.

The man pages for 4.3BSD and older versions of *tgetent()* instruct the user to allocate a 1K buffer for the termcap entry. The entry gets null-terminated by the termcap library, so that makes the maximum safe length for a termcap entry 1k-1 (1023) bytes. Depending on what the application and the termcap library being used do, and where in the termcap file the terminal type that *tgetent()* is searching for is, several bad things can happen.

Some termcap libraries print a warning message or exit if they find an entry that's longer than 1023 bytes; others don't; others truncate the entries to 1023 bytes. Some application programs allocate more than the recommended 1K for the termcap entry; others don't.

Each termcap entry has two important sizes associated with it: before "tc" expansion, and after "tc" expansion. "tc" is the capability that tacks on another termcap entry to the end of the current one, to add on its capabilities. If a termcap entry doesn't use the "tc" capability, then of course the two lengths are the same.

The "before tc expansion" length is the most important one, because it affects more than just users of that particular terminal. This is the length of the entry as it exists in **\$OPENNT_ROOT/etc/termcap**, minus the backslash-newline pairs, which *tgetent()* strips out while reading it. Some termcap libraries also strip off the final newline (GNU termcap does not). Now suppose:

- a termcap entry before expansion is more than 1023 bytes long,
- and the application has only allocated a 1k buffer,
- and the termcap library (like the one in BSD/OS 1.1 and GNU) reads the whole entry into the buffer, no matter what its length, to see if it's the entry it wants,
- and *tgetent()* is searching for a terminal type that either is the long entry, appears in the termcap file after the long entry, or doesn't appear in the file at all (so that *tgetent()* has to search the whole termcap file).

Then *tgetent()* will overwrite memory, perhaps its stack, and probably core dump the program. Programs like telnet are particularly vulnerable; modern telnets pass along values like the terminal type automatically. The results are almost as undesirable with a termcap library, like SunOS 4.1.3 and Ultrix 4.4, that prints warning messages when it reads an overly long termcap entry. If a termcap library truncates long entries, like OSF/1 3.0, it is immune to dying here but will return incorrect data for the terminal.

The "after tc expansion" length will have a similar effect to the above, but only for people who actually set *TERM* to that terminal type, since *tgetent()* only does "tc" expansion once it's found the terminal type it was looking for, not while searching.

In summary, a termcap entry that is longer than 1023 bytes can cause (on various combinations of termcap libraries and applications) a core dump, warnings, or incorrect operation. If it's too long even before "tc" expansion, it will have this effect even for users of some other terminal types and users whose *TERM* variable does not have a termcap entry.

When in **-C** (translate to termcap) mode, the ncurses implementation of *tic* issues warning messages when the pre-tc length of a termcap translation is too long. The **-c** (check) option also checks resolved (after tc expansion) lengths.

Binary Compatibility

It is not wise to count on portability of binary terminfo entries between commercial systems. The problem is that there are at least two versions of terminfo (under HP-UX and AIX) which diverged from System V terminfo after SVr1, and have added extension capabilities to the string table that (in the binary format) collide with System V and XSI Curses extensions.

EXTENSIONS

The **%x** operator in parameterized strings is unique to the ncurses implementation of **tparm** (it is required in order to support an unfortunate choice of **initc** format on the Linux console).

Some SVr4 curses implementations, and all previous to SVr4, don't interpret the **%A** and **%O** operators in parameter strings.

SVr4/XPG4 do not specify whether **msgr** licenses movement while in an alternate-character-set mode (such modes may, among other things, map CR and NL to characters that don't trigger local motions). The ncurses implementation ignores **msgr** in ALTCHARSET mode. This raises the possibility that an XPG4 implementation making the opposite interpretation may need terminfo entries made for ncurses to have **msgr** turned off.

The ncurses library handles insert-character and insert-character modes in a slightly non-standard way in order to get better update efficiency. See the "Insert/Delete Character" subsection above.

The parameter substitutions for **set_clock** and **display_clock** are not documented in SVr4 or the XSI Curses standard. They are deduced from the documentation for the AT&T 505 terminal.

Be careful assigning the **kmous** capability. The ncurses wants to interpret it as KEY_MOUSE, for use by terminals and emulators like xterm that can return mouse-tracking information in the keyboard-input stream.

Different commercial ports of terminfo and curses support different subsets of the XSI Curses standard and (in some cases) different extension sets. Here is a summary, accurate as of October 1995:

SVR4, Solaris, ncurses

These support all SVr4 capabilities.

SGI Supports the SVr4 set, adds one undocumented extended string capability (**set_pglen**).

SVr1, Ultrix

These support a restricted subset of terminfo capabilities. The booleans end with **xon_xoff**; the numerics with **width_status_line**; and the strings with **prtr_non**.

HP/UX Supports the SVr1 subset, plus the SVr[234] numerics **num_labels**, **label_height**, **label_width**, plus function keys 11 through 63, plus **plab_norm**, **label_on**, and **label_off**, plus some incompatible extensions in the string table.

AIX Supports the SVr1 subset, plus function keys 11 through 63, plus a number of incompatible string table extensions.

OSF Supports both the SVr4 set and the AIX extensions.

FILES

The terminfo packages makes use of the following files:

terminfo/?/*

Files containing terminal descriptions.

SEE ALSO

tic, curses

printf

term

AUTHORS

Zeyd M. Ben-Halim, Eric S. Raymond. Descends from the original pcurses by Pavel Curtis.

NAME

termios – general terminal line discipline

SYNOPSIS

```
#include <termios.h>
```

DESCRIPTION

This describes a general terminal line discipline that is supported on tty asynchronous communication ports.

Opening a Terminal Device File

When a terminal file is opened, it normally causes the process to wait until a connection is established. For most hardware, the presence of a connection is indicated by the assertion of the hardware CARRIER line. In practice, applications and become an application's standard input, output, and error files.

Job Control in a Nutshell

Every process is associated with a particular process group and session. The grouping is hierarchical: every member of a particular process group is a member of the same session. This structuring is used in managing groups of related processes for purposes of that is, the ability from the keyboard (or from program control) to simultaneously stop or restart a complex command (a command composed of one or more related processes). The grouping into process groups allows delivering of signals that stop or start the group as a whole, along with arbitrating which process group has access to the single controlling terminal. The grouping at a higher layer into sessions is to restrict the job control related signals and system calls to within processes resulting from a particular instance of a "login". Typically, a session is created when a user logs in, and the login terminal is setup to be the controlling terminal; all processes spawned from that login shell are in the same session, and inherit the controlling terminal. A job control shell operating interactively (that is, reading commands from a terminal) normally groups related processes together by placing them into the same process group. A set of processes in the same process group is collectively referred to as a "job". When the foreground process group of the terminal is the same as the process group of a particular job, that job is said to be in the "foreground". When the process group of the terminal is different than the process group of a job (but is still the controlling terminal), that job is said to be in the "background". Normally the shell reads a command and starts the job that implements that command. If the command is to be started in the foreground (typical), it sets the process group of the terminal to the process group of the started job, waits for the job to complete, and then sets the process group of the terminal back to its own process group (it puts itself into the foreground). If the job is to be started in the background (as denoted by the shell operator "&"), it never changes the process group of the terminal and doesn't wait for the job to complete (that is, it immediately attempts to read the next command). If the job is started in the foreground, the user may type a key (usually ^Z) which generates the terminal stop signal (SIGTSTP) and has the affect of stopping the entire job. The shell will notice that the job stopped, and will resume running after placing itself in the

foreground. The shell also has commands for placing stopped jobs in the background, and for placing stopped or background jobs into the foreground.

Orphaned Process Groups

An orphaned process group is a process group that has no process whose parent is in a different process group, yet is in the same session. Conceptually it means a process group that doesn't have a parent that could do anything if it were to be stopped. For example, the initial login shell is typically in an orphaned process group. Orphaned process groups are immune to keyboard generated stop signals and job control signals resulting from reads or writes to the controlling terminal.

The Controlling Terminal

A terminal may belong to a process as its controlling terminal. Each process of a session that has a controlling terminal has the same controlling terminal. A terminal may be the controlling terminal for at most one session. The controlling terminal for a session is allocated by the session leader by issuing an *open()* on a terminal that is not already a controlling terminal for another session. A controlling terminal is never acquired by merely opening a terminal device file. When a controlling terminal becomes associated with a session, its foreground process group is set to the process group of the session leader.

The controlling terminal is inherited by a child process during a *fork()* function call. A process relinquishes its controlling terminal when it creates a new session with the function; other processes remaining in the old session that had this terminal as their controlling terminal continue to have it. A process does not relinquish its controlling terminal simply by closing all of its file descriptors associated with the controlling terminal if other processes continue to have it open.

When a controlling process terminates, the controlling terminal is disassociated from the current session, allowing it to be acquired by a new session leader. Subsequent access to the terminal by other processes in the earlier session will be denied, with attempts to access the terminal treated as if modem disconnect had been sensed.

Terminal Access Control

If a process is in the foreground process group of its controlling terminal, read operations are allowed. Any attempts by a process in a background process group to read from its controlling terminal causes a SIGTTIN signal to be sent to the process's group unless one of the following special cases apply: If the reading process is ignoring or blocking the SIGTTIN signal, or if the process group of the reading process is orphaned, the *read()* returns -1 with *errno* set to [EIO] and no signal is sent. The default action of the SIGTTIN signal is to stop the process to which it is sent.

If a process is in the foreground process group of its controlling terminal, write operations are allowed. Attempts by a process in a background process group to write to its controlling terminal will cause the process group to be sent a SIGTTOU signal unless one of the following special cases apply: If TOSTOP is not set, or if TOSTOP is set and the process is ignoring or blocking the

SIGTTOU signal, the process is allowed to write to the terminal and the SIGTTOU signal is not sent. If TOSTOP is set, and the process group of the writing process is orphaned, and the writing process is not ignoring or blocking SIGTTOU, the *write* returns -1 with *errno* set to [EIO] and no signal is sent.

Certain calls that set terminal parameters are treated in the same fashion as *write*, except that TOSTOP is ignored; that is, the effect is identical to that of terminal writes when TOSTOP is set.

Input Processing and Reading Data

A terminal device associated with a terminal device file may operate in full-duplex mode, so that data may arrive even while output is occurring. Each terminal device file has associated with it an input queue, into which incoming data is stored by the system before being read by a process. The system imposes a limit, {MAX_INPUT}, on the number of bytes that may be stored in the input queue. The behavior of the system when this limit is exceeded depends on the setting of the IMAXBEL flag in the termios **c_iflag**. If this flag is set, the terminal is sent an ASCII BEL character each time a character is received while the input queue is full. Otherwise, the input queue is flushed upon receiving the character.

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode or noncanonical mode. Additionally, input characters are processed according to the **c_iflag** and **c_lflag** fields. Such processing can include echoing, which in general means transmitting input characters immediately back to the terminal when they are received from the terminal. This is useful for terminals that can operate in full-duplex mode.

The manner in which data is provided to a process reading from a terminal device file is dependent on whether the terminal device file is in canonical or noncanonical mode.

Another dependency is whether the O_NONBLOCK flag is set by *open()* or *fcntl()*. If the O_NONBLOCK flag is clear, then the read request is blocked until data is available or a signal has been received. If the O_NONBLOCK flag is set, then the read request is completed, without blocking, in one of three ways:

1. If there is enough data available to satisfy the entire request, and the read completes successfully the number of bytes read is returned.
2. If there is not enough data available to satisfy the entire request, and the read completes successfully, having read as much data as possible, the number of bytes read is returned.
3. If there is no data available, the read returns -1, with *errno* set to [EAGAIN.]

When data is available depends on whether the input processing mode is canonical or noncanonical.

Canonical Mode Input Processing

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a newline `\n` character, an end-of-file (EOF) character, or an end-of-line (EOL) character. See the “Special Characters” section for more information on EOF and EOL. This means that a read request will not return until an entire line has been typed, or a signal has been received. Also, no matter how many bytes are requested in the read call, at most one line is returned. It is not, however, necessary to read a whole line at once; any number of bytes, even one, may be requested in a read without losing information.

`{MAX_CANON}` is a limit on the number of bytes in a line. The behavior of the system when this limit is exceeded is the same as when the input queue limit `{MAX_INPUT}` is exceeded.

Erase and kill processing occur when either of two special characters, the ERASE and KILL characters (see the “Special Characters section”), is received. This processing affects data in the input queue that has not yet been delimited by a newline NL, EOF, or EOL character. This un-delimited data makes up the current line. The ERASE character deletes the last character in the current line, if there is any. The KILL character deletes all data in the current line, if there is any. The ERASE and KILL characters have no effect if there is no data in the current line. The ERASE and KILL characters themselves are not placed in the input queue.

Noncanonical Mode Input Processing

In noncanonical mode input processing, input bytes are not assembled into lines, and erase and kill processing does not occur. The values of the MIN and TIME members of the `c_cc` array are used to determine how to process the bytes received.

MIN represents the minimum number of bytes that should be received when the `read` function successfully returns. TIME is a timer of 0.1 second granularity that is used to time out bursty and short term data transmissions. If MIN is greater than `{MAX_INPUT}` the response to the request is undefined. The four possible values for MIN and TIME and their interactions are described below.

Case A: MIN > 0, TIME > 0

In this case TIME serves as an inter-byte timer and is activated after the first byte is received. Since it is an inter-byte timer, it is reset after a byte is received. The interaction between MIN and TIME is as follows: as soon as one byte is received, the inter-byte timer is started. If MIN bytes are received before the inter-byte timer expires (remember that the timer is reset upon receipt of each byte), the read is satisfied. If the timer expires before MIN bytes are received, the characters received to that point are returned to the user. Note that if TIME expires at least one byte is returned because the timer would not have been enabled unless a byte was received. In this case (MIN > 0, TIME > 0) the read blocks until the MIN and TIME mechanisms are activated by the receipt of the first byte, or a signal is received. If data is in the buffer at the time of the `read()`, the result is as if data had been received immediately after the `read()`.

Case B: MIN > 0, TIME = 0

In this case, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read is not satisfied until MIN bytes are received (i.e., the pending read blocks until MIN bytes are received), or a signal is received. A program that uses this case to read record-based terminal I/O may block indefinitely in the read operation.

Case C: MIN = 0, TIME > 0

In this case, since MIN = 0, TIME no longer represents an inter-byte timer. It now serves as a read timer that is activated as soon as the read function is processed. A read is satisfied as soon as a single byte is received or the read timer expires. Note that in this case if the timer expires, no bytes are returned. If the timer does not expire, the only way the read can be satisfied is if a byte is received. In this case the read will not block indefinitely waiting for a byte; if no byte is received within TIME*0.1 seconds after the read is initiated, the read returns a value of zero, having read no data. If data is in the buffer at the time of the read, the timer is started as if data had been received immediately after the read.

Case D: MIN = 0, TIME = 0

The minimum of either the number of bytes requested or the number of bytes currently available is returned without waiting for more bytes to be input. If no characters are available, read returns a value of zero, having read no data.

Writing Data and Output Processing

When a process writes one or more bytes to a terminal device file, they are processed according to the **c_oflag** field (see the “Output Modes” section). The implementation may provide a buffering mechanism; as such, when a call to *write()* completes, all of the bytes written have been scheduled for transmission to the device, but the transmission will not necessarily have been completed.

Special Characters

Certain characters have special functions on input or output or both. These functions are summarized as follows:

- INTR** Special character on input and is recognized if the ISIG flag (see the “Local Modes” section) is enabled. Generates a SIGINT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the INTR character is discarded when processed.
- QUIT** Special character on input and is recognized if the ISIG flag is enabled. Generates a SIGQUIT signal which is sent to all processes in the foreground process group for which the terminal is the controlling terminal. If ISIG is set, the QUIT character is discarded when processed.
- ERASE** Special character on input and is recognized if the ICANON flag is set. Erases the last character in the current line; see “Canonical Mode Input Processing”. It does not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character. If ICANON is set, the ERASE character is discarded when processed.
- KILL** Special character on input and is recognized if the ICANON flag is set.

	Deletes the entire line, as delimited by a NL, EOF, or EOL character. If ICANON is set, the KILL character is discarded when processed.
EOF	Special character on input and is recognized if the ICANON flag is set. When received, all the bytes waiting to be read are immediately passed to the process, without waiting for a newline, and the EOF is discarded. Thus, if there are no bytes waiting (that is, the EOF occurred at the beginning of a line), a byte count of zero is returned from the <i>read()</i> , representing an end-of-file indication. If ICANON is set, the EOF character is discarded when processed. NL Special character on input and is recognized if the ICANON flag is set. It is the line delimiter \n.
EOL	Special character on input and is recognized if the ICANON flag is set. Is an additional line delimiter, like NL.
SUSP	If the ISIG flag is enabled, receipt of the SUSP character causes a SIGTSTP signal to be sent to all processes in the foreground process group for which the terminal is the controlling terminal, and the SUSP character is discarded when processed.
STOP	Special character on both input and output and is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to temporarily suspend output. It is useful with fast terminals to prevent output from disappearing before it can be read. If IXON is set, the STOP character is discarded when processed.
START	Special character on both input and output and is recognized if the IXON (output control) or IXOFF (input control) flag is set. Can be used to resume output that has been suspended by a STOP character. If IXON is set, the START character is discarded when processed. CR Special character on input and is recognized if the ICANON flag is set; it is the \r, as denoted in the C Standard. When ICANON and ICRNL are set and IGNCR is not set, this character is translated into a NL, and has the same effect as a NL character.
VBELTIME	The value of VBELTIME specifies the minimum period of silence between bell sounds from the terminal, in tenths of a second. After a bell sounds at the terminal, the system ignores any other bell signals delivered to the terminal for the next VBELTIME tenths of seconds. All other characters are delivered as they would normally be.

The NL and CR characters cannot be changed. The values for all the remaining characters can be set and are described later in the document under Special Control Characters.

Special character functions associated with changeable special control characters can be disabled individually by setting their value to `{_POSIX_VDISABLE}`; see “Special Control Characters”.

If two or more special characters have the same value, the function performed when that character is received is undefined.

Modem Disconnect

If a modem disconnect is detected by the terminal interface for a controlling terminal, and if CLOCAL is not set in the **c_cflag** field for the terminal, the SIGHUP signal is sent to the controlling process associated with the terminal. Unless other arrangements have been made, this causes the controlling process to terminate. Any subsequent call to the *read()* function returns the value zero, indicating end of file. Thus, processes that read a terminal file and test for end-of-file can terminate appropriately after a disconnect. Any subsequent *write()* to the terminal device returns -1, with *errno* set to [EIO,] until the device is closed.

General Terminal Interface**Closing a Terminal Device File**

The last process to close a terminal device file causes any output to be sent to the device and any input to be discarded. Then, if HUPCL is set in the control modes, and the communications port supports a disconnect function, the terminal device performs a disconnect.

Parameters That Can Be Set

Routines that need to control certain terminal I/O characteristics do so by using the termios structure as defined in the header **<termios.h>**. This structure contains minimally four scalar elements of bit flags and one array of special characters. The scalar flag elements are named: **c_iflag**, **c_oflag**, **c_cflag**, and **c_lflag**. The character array is named **c_cc**, and its maximum index is NCCS.

Input Modes

Values of the **c_iflag** field describe the basic terminal input control, and are composed of following masks:

IGNBRK	Ignore BREAK condition.
BRKINT	Map BREAK to SIGINTR.
IGNPAR	Ignore (discard) parity errors.
PARMRK	Mark parity and framing errors.
INPCK	Enable checking of parity errors.
ISTRIP	Strip 8th bit off chars.
INLCR	Map NL into CR
IGNCR	Ignore CR
ICRNL	Map CR to NL (ala CRMOD)
IXON	enable output flow control
IXOFF	Enable input flow control
IMAXBEL	Ring bell on input queue full

In the context of asynchronous serial data transmission, a break condition is defined as a sequence of zero-valued bits that continues for more than the time to send one byte. The entire sequence of zero-valued bits is interpreted as a single break condition, even if it continues for a time equivalent to more than one byte. In contexts other than asynchronous serial data transmission the definition of a break condition is implementation defined.

If IGNBRK is set, a break condition detected on input is ignored, that is, not put on the input queue and therefore not read by any process. If IGNBRK is not set and BRKINT is set, the break condition flushes the input and output queues and if the terminal is the controlling terminal of a foreground process group, the break condition generates a single SIGINT signal to that foreground process group. If neither IGNBRK nor BRKINT is set, a break condition is read as a single `\0`, or if PARMRK is set, as `\377, \0, \0`.

If IGNPAR is set, a byte with a framing or parity error (other than break) is ignored.

If PARMRK is set, and IGNPAR is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence `\377, \0, X`, where `\377, \0` is a two-character flag preceding each sequence and `X` is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of `\377` is given to the application as `\377, \377`. If neither PARMRK nor IGNPAR is set, a framing or parity error (other than break) is given to the application as a single character `\0`.

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled, allowing output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled (see “Control Modes”). If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected recognizes the parity bit, but the terminal special file does not check whether this bit is set correctly or not.

If ISTRIP is set, valid input bytes are first stripped to seven bits, otherwise all eight bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). If IGNCR is not set and ICRNL is set, a received CR character is translated into a NL character.

If IXON is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. If IXANY is also set, then any character may restart output. When IXON is set, START and STOP characters are not read, but merely perform flow control functions. When IXON is not set, the START and STOP characters are read.

If IXOFF is set, start/stop input control is enabled. The system shall transmit one or more STOP characters, which are intended to cause the terminal device to stop transmitting data, as needed to prevent the input queue from overflowing and causing the undefined behavior described in “Input Processing and Reading Data”, and shall transmit one or more START characters, which are intended to cause the terminal device to resume transmitting data, as soon as the device can continue transmitting data without risk of overflowing the input queue. The precise conditions under which STOP and START characters are transmitted are implementation defined.

If IMAXBEL is set and the input queue is full, subsequent input shall cause an ASCII BEL character to be transmitted to the the output queue.

The initial input control value after `open()` is implementation defined.

Output Modes

Values of the **c_oflag** field describe the basic terminal output control, and are composed of the following masks:

OPOST Enable following output processing.

ONLCR

Map NL to CR-NL (a la CRMOD).

ONLRET

NL performs the ASCII CR function.

ONOCR

No ASCII CR output in column 0.

If OPOST is set, the remaining flag masks are interpreted as follows; otherwise characters are transmitted without change.

If ONLCR is set, newlines are translated to carriage return, linefeeds.

Control Modes

Values of the **c_cflag** field describe the basic terminal hardware control, and are composed of the following masks. Not all values specified are supported by all hardware.

CSIZE Character size mask.

CS5 5 bits (pseudo).

CS6 6 bits.

CS7 7 bits.

CS8 8 bits.

CSTOPB Send 2 stop bits.

CREAD Enable receiver.

PARENB Parity enable.

PARODD Odd parity, else even.

HUPCL Hang up on last close.

CLOCAL Ignore modem status lines.

The CSIZE bits specify the byte size in bits for both transmission and reception. The **c_cflag** is masked with CSIZE and compared with the values CS5, CS6, CS7, or CS8. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stop bits are normally used.

If CREAD is set, the receiver is enabled. Otherwise, no character is received. Not all hardware supports this bit. In fact, this flag is pretty silly and if it were not part of the *termios* specification it would be omitted.

If PARENB is set, parity generation and detection are enabled and a parity bit is added to each character. If parity is enabled, PARODD specifies odd parity if set, otherwise even parity is used.

If HUPCL is set, the modem control lines for the port are lowered when the last process with the port open closes the port or the process terminates. The modem connection is broken.

If CLOCAL is set, a connection does not depend on the state of the modem status lines. If CLOCAL is clear, the modem status lines are monitored.

Under normal circumstances, a call to the `open()` function waits for the modem connection to complete. However, if the `O_NONBLOCK` flag is set or if `CLOCAL` has been set, the `open()` function returns immediately without waiting for the connection.

If the object for which the control modes are set is not an asynchronous serial connection, some of the modes may be ignored; for example, if an attempt is made to set the baud rate on a network connection to a terminal on another host, the baud rate may or may not be set on the connection between that terminal and the machine it is directly connected to.

Local Modes

Values of the `c_lflag` field describe the control of various functions, and are composed of the following masks.

ECHOK	Visual erase for line kill.
ECHOE	Visually erase characters.
ECHO	Enable echoing.
ECHONL	Echo NL even if ECHO is off.
ISIG	Enable signals INTR, QUIT, [D]SUSP. */
ICANON	Canonicalize input lines.
IEXTEN	Enable DISCARD and LNEXT.
EXTPROC	External processing; in external processing mode, characters are never echoed, special characters are never processed, and no characters are erased. This mode is used in programs such as <i>telnetd</i> .
TOSTOP	Stop background jobs from output.

If ECHO is set, input characters are echoed back to the terminal. If ECHO is not set, input characters are not echoed.

If ECHOE and ICANON are set, the ERASE character causes the terminal to erase the last character in the current line from the display, if possible. If there is no character to erase, an implementation may echo an indication that this was the case or do nothing.

If ECHOK and ICANON are set, the KILL character causes the current line to be discarded and the system echoes the `\n` character after the KILL character.

If ECHONL and ICANON are set, the `\n` character echoes even if ECHO is not set.

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL, as described in “Canonical Mode Input Processing”.

If ICANON is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least MIN bytes have been received or the timeout value TIME expired between bytes. The time value represents tenths

of seconds. See “Noncanonical Mode Input Processing” for more details.

If ISIG is set, each input character is checked against the special control characters INTR, QUIT, and SUSP (job control only). If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set.

If IEXTEN is set, implementation-defined functions are recognized from the input data. How IEXTEN being set interacts with ICANON, ISIG, IXON, or IXOFF is implementation defined. If IEXTEN is not set, then implementation-defined functions are not recognized, and the corresponding input characters are not processed as described for ICANON, ISIG, IXON, and IXOFF.

If NOFLSH is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters are not be done.

If TOSTOP is set, the signal SIGTTOU is sent to the process group of a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal, by default, stops the members of the process group. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring SIGTTOU signals are excepted and allowed to produce output and the SIGTTOU signal is not sent.

Special Control Characters

The special control characters values are defined by the array **c_cc**. This table lists the array index, the corresponding special character, and the system default value.

Index Name	Special Character	Default Value
VEOF	EOF	^D
VEOL	EOL	_POSIX_VDISABLE
VERASE	ERASE	^H
VKILL	KILL	^U
VINTR	INTR	^C
VQUIT	QUIT	^\
VSUSP	SUSP	^Z
VSTART	START	^Q
VSTOP	STOP	^S
VMIN	---	1
VTIME	---	0
VBELTIME	---	3

If the value of one of the changeable special control characters (see “Special Characters”) is {_POSIX_VDISABLE}, that function is disabled; that is, no input data is recognized as the disabled special character. If ICANON is not set, the value of {_POSIX_VDISABLE} has no special meaning for the VMIN and VTIME entries of the **c_cc** array.

NAME

tty – general terminal interface

SYNOPSIS

```
#include <sys/ioctl.h>
```

DESCRIPTION

This section describes the interface to the terminal drivers in the system.

Terminal Special Files

Each terminal on the system usually has a terminal special device file associated with it in the directory **/dev/** (for example, **/dev/ttyn02**). When a user logs into the system on one of these terminals, the system has already opened the associated device and prepared the line for normal interactive use. There is also a special case of a terminal file that connects not to a hardware terminal port, but to another program on the other side. These special terminal devices are called *ptys* and provide the mechanism necessary to give users the same interface to the system when logging in over a network (using *telnet* for example.) Even in these cases the details of how the terminal file was opened and set up is already handled by special software in the system. Thus, users do not normally need to worry about the details of how these lines are opened or used.

When an interactive user logs in, the system prepares the line to behave in a certain way (called a *line discipline*), the particular details of which is described in *stty* at the command level, and in *termios* at the programming level. A user may be concerned with changing settings associated with his particular login terminal and should refer to the preceding man pages for the common cases. The remainder of this man page is concerned with describing details of using and controlling terminal devices at a low level, such as that possibly required by a program wishing to provide features similar to those provided by the system.

Line disciplines

A terminal file is used like any other file in the system in that it can be opened, read, and written to using standard system calls. For each existing terminal file, there is a software processing module called a *line discipline* is associated with it. The *line discipline* essentially glues the low level device driver code with the high level generic interface routines (such as *read()* and *write()*), and is responsible for implementing the semantics associated with the device. When a terminal file is first opened by a program, the default *line discipline* called the *termios* line discipline is associated with the file. This is the primary line discipline that is used in most cases and provides the semantics that users normally associate with a terminal. When the *termios* line discipline is in effect, the terminal file behaves and is operated according to the rules described in *termios*. Please refer to that man page for a full description of the terminal semantics. The operations described here generally represent features common across all *line disciplines*, however some of these calls may not make sense in conjunction with a line discipline other than *termios*, and some may not be supported by the underlying hardware (or lack thereof, as in the case of *ptys*).

Terminal File Operations

The majority of terminal settings can be handled through the *tc** family of interfaces. *Termios* defines them as function calls, not *ioctl()* requests. The following section lists the available *ioctl()* requests. The name of the request, a description of its purpose, and the typed *argp* parameter (if any) are listed. For example, the first entry says

TIOCGWINSZ

and would be called on the terminal associated with file descriptor zero by the following code fragment:

```
struct winsize ws;
ioctl(0, TIOCGWINSZ, &ws);
```

Terminal File Request Descriptions

TIOCSSTI char **cp*

Simulate typed input. Pretend as if the terminal received the character pointed to by *cp*.

TIOCSTOP void

Stop output on the terminal (like typing ^S at the keyboard).

TIOCSTART void

Start output on the terminal (like typing ^Q at the keyboard).

TIOCGWINSZ struct winsize **ws*

Put the window size information associated with the terminal in the **winsize** structure pointed to by *ws*. The window size structure contains the number of rows and columns (and pixels if appropriate) of the devices attached to the terminal. It is set by user software and is the means by which most full-screen oriented programs determine the screen size. The **winsize** structure is defined in **<sys/ioctl.h>**.

TIOCSWINSZ struct winsize **ws*

Set the window size associated with the terminal to be the value in the **winsize** structure pointed to by *ws* (see above).

SEE ALSO

stty

fcntl(), *ioctl()*, *tcgetattr()*, *tcsetattr()*

pty, *termios*