

QNX4.0

Utilities Reference



QNX[®] 4.0 Operating System

Utilities Reference

Quantum Software Systems Ltd.

175 Terrence Matthews Crescent
Kanata, Ontario K2M 1W8
Canada
(613) 591-0931

© Quantum Software Systems Ltd. 1991
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise without the prior written permission of Quantum Software Systems Ltd.

Although every precaution has been taken in the preparation of this book, we assume no responsibility for any errors or omissions, nor do we assume liability for damages resulting from the use of the information contained in this book.

This edition: April, 1991

GNX is a registered trademark of Quantum Software Systems Ltd.
UNIX is a registered trademark of AT&T.

UNIX Software Acknowledgments and Copyrights

The following individuals and organizations have placed the utilities listed below in the public domain. We have enhanced some of these utilities and hope you will find them helpful. The full source code for these utilities along with additional notes and credits are available from our on-line update and conferencing system QUICS:

Steve Kirkendall (**ctags**); Mark Nudelman (**less**); Greg Yachuk (**make**); The University of California at Berkeley (**flex** [**lex**] and **yacc**).

Flex © 1987 The University of California

The **gawk** utility is the Free Software Foundation's version of UNIX **awk**.
Written by Paul Rubin, August 1986.
© 1986, 1988, 1989 the Free Software Foundation, Inc.

The **gawk** utility is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 1, or (at your option) any later version.

The full source code for **gawk** as well as a copy of the GNU General Public License can be found in electronic form on QUICS. You can also write to:

The Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

Cover art by Robert Tinney.

Printed in Canada.

Acknowledgments

Several members of Quantum Software Systems Ltd. have contributed to the design and implementation of QNX 4.0. This product embodies more than a decade of research, development, and field-tested experience. Veteran QNX users will recognize the performance advantages and comfortable feel of their familiar OS, while new users can expect a “standard” operating system with a difference. We are proud to see the QNX operating system enter the standards arena without sacrificing the fundamental qualities of its original vision.

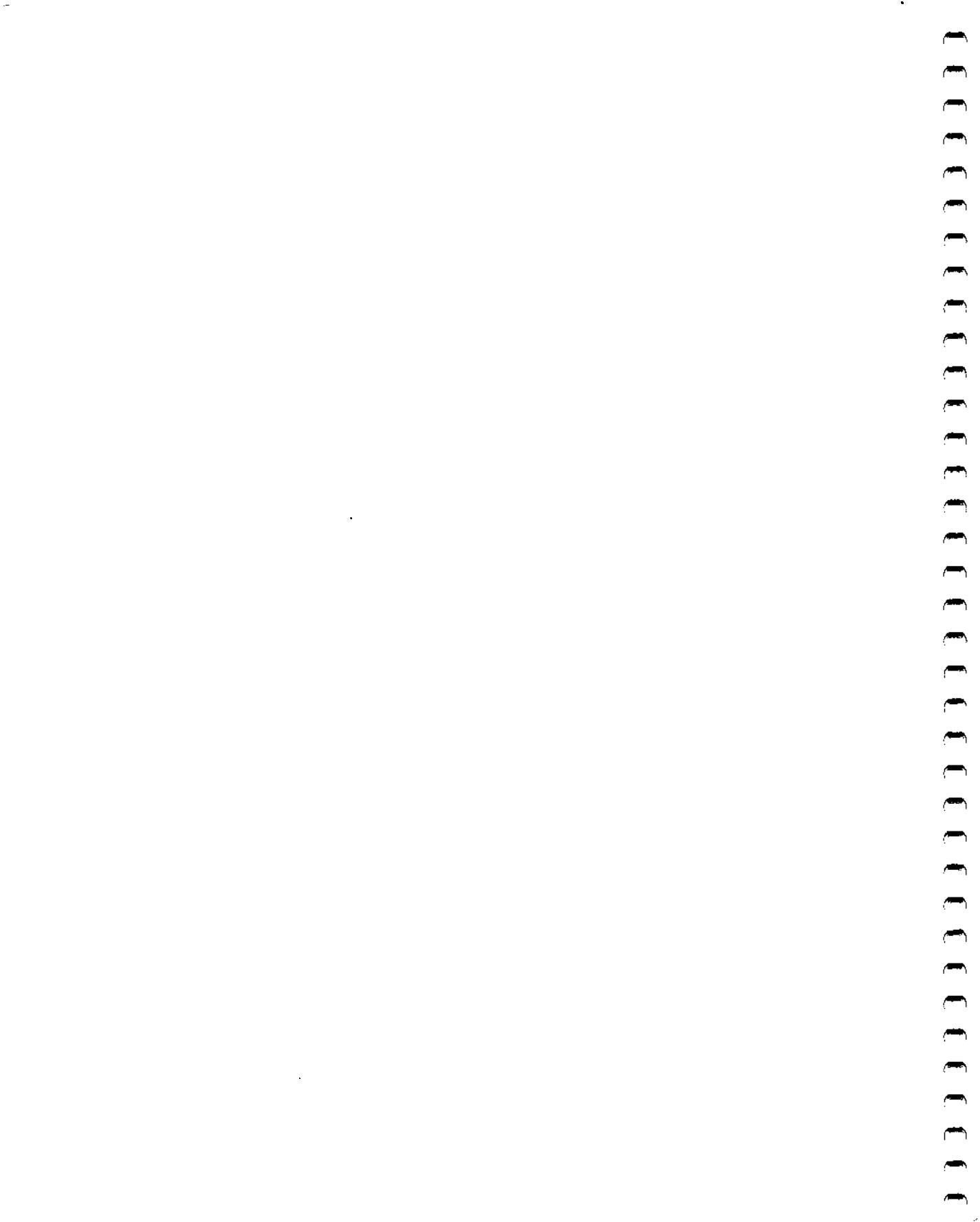
The following people deserve a special word of thanks and acknowledgment for their substantial contribution to QNX 4.0:

Software Development:

Luc N. Bazinet
Gordon Bell
Andrew C. Boyd
Brian Campbell
Grant D. Courville
Dan Dodge
Andrew Edgar
William A. Flowers
Dan Frank
Jeffrey G. George
Dan Hildebrand
Eric Johnson
Debbie Kennedy
Terry Laughlin
Steve McPolin
Rob Oakley
Martin Walter

Documentation:

Kevin Didsbury
Paul N. Leroux
Paul McManus
John M. Ostrander



About this Guide

What will I find in this guide?

The QNX operating system *Utilities Reference* accompanies the QNX operating system (release 4.0) and is intended for both application developers and end-users.



We assume that QNX is already installed and running on your machine. If not, you—or your system administrator—should refer to the *Administrator's Guide* for installation and configuration.

The *Utilities Reference* contains the following:

- an alphabetical list of all the utilities (p. ix)
- a “man page” for each utility, arranged alphabetically (p. 3)

Note to system administrators

This guide doesn't contain all the utilities you might use for system administration. Those utilities, along with drivers and system managers, are documented in the *Administrator's Guide*.

Conventions used in this guide

Technical terms

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications. The following table summarizes our conventions.

Reference	Example
command names	ls
data types	<i>long</i>
environment variables	PATH
error numbers	[EINTR]
filenames	/dev/tty
functions	<i>system()</i>
function arguments	<i>arg1</i>
global externals	<i>errno</i>
headers	<sys/stat.h>
options	-c
option arguments	-w <i>width</i>
operands	<i>file_name</i>
parameters	<directory pathname>
symbolic constants	{_POSIX_VDISABLE}

How keys are displayed in print

When this guide instructs you to press keys on your keyboard, the keys you're to type appear between angle brackets. For example, if a passage instructs you to:

press <Enter>

you press the key labeled Enter.

Key chords

When this guide instructs you to press more than one key at the same time, the keys are separated by a hyphen (-). For example, if you see:

press <Ctrl>-<N>

you would:

- 1) press <Ctrl> and hold it down
- 2) press <N>
- 3) quickly release both keys.

Entering text

When this guide instructs you to enter text from your keyboard, the text you should type appears in a **constant-width typeface**. For example, if you see:

Type **pwd**

you would type:

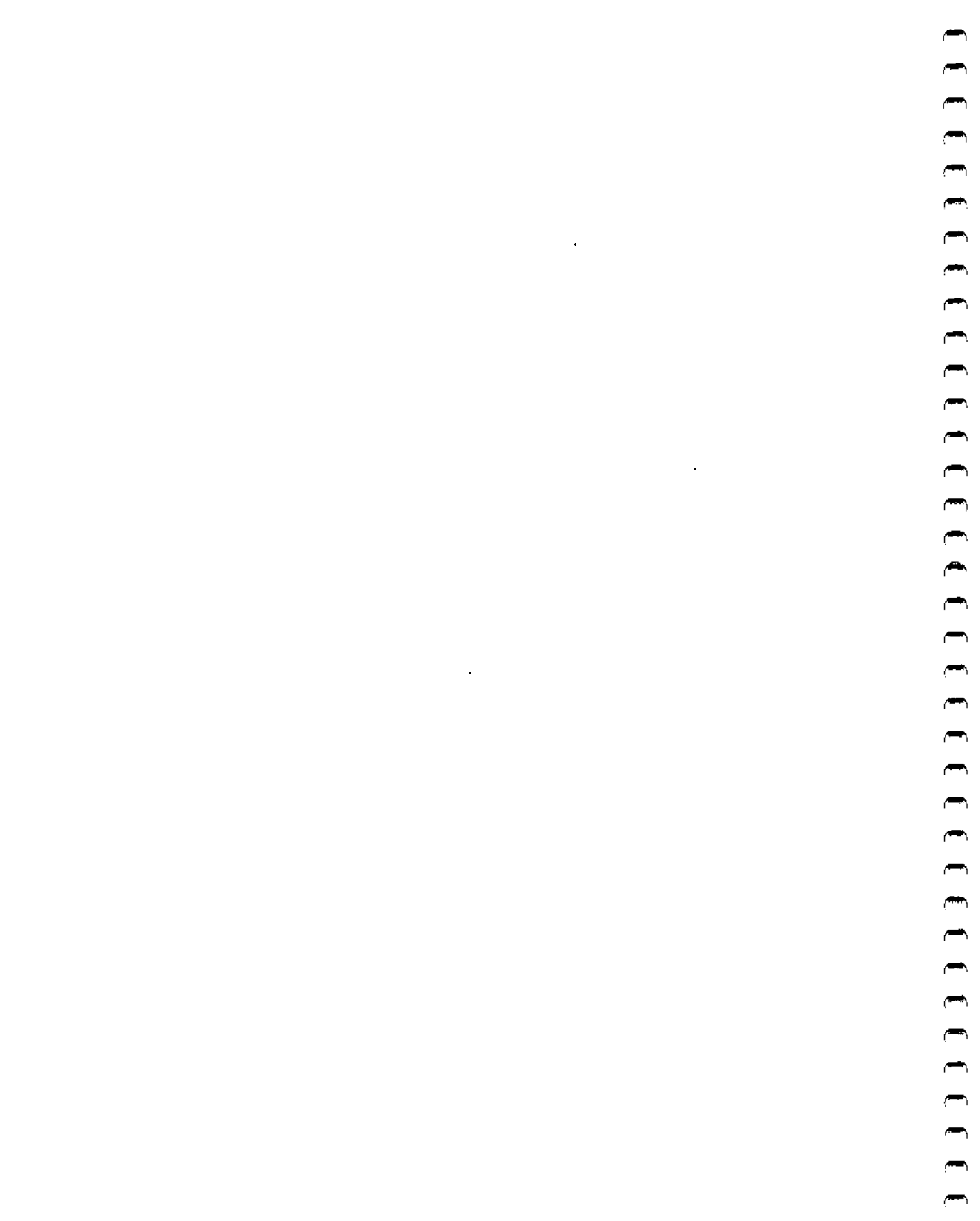
pwd

Important notes

Important text is indicated by a pointing hand (☞). For example:



If you want to move a file from another user's directory, you must first have the appropriate permissions on that directory.



Summary of Utilities

awk	pattern scanning and processing language (POSIX) 3
basename	return nondirectory portion of pathname (POSIX) 23
cat	concatenate and print files (POSIX) 25
cc	compile command (QNX) 27
cd	change working directory (POSIX) 37
chgrp	change file group ownership (POSIX) 41
chkfsys	check an entire filesystem for consistency (QNX) 43
chmod	change file modes (POSIX) 49
chown	change file ownership (POSIX) 53
cksum	display file checksums and block counts (POSIX) 55
clock	display date and time on console (QNX) 57
cmp	compare two files (POSIX) 59
cp	copy files (POSIX) 61
cron	clock server (QNX) 73
crontab	schedule periodic background work (POSIX) 75
ctags	generate tags files (UNIX) 79
date	display or set the date and time (POSIX) 81
dcheck	check a disk for bad blocks (QNX) 85
df	report disk space free (POSIX) 89

diff	report the differences between two files (POSIX) 91
dinit	disk initialization (QNX) 97
dirname	return directory portion of pathname (POSIX) 99
du	estimate disk space usage (POSIX) 101
dumper	dump postmortem state of a program (QNX) 103
echo	write arguments to standard output (POSIX) 105
fdformat	format floppy diskettes (QNX) 107
find	find files (POSIX) 111
grep	search for string patterns (POSIX) 119
hd	display files in decimal, hex, octal, or ASCII (QNX) 125
head	copy the first part of files (POSIX) 129
infocmp	output the contents of a terminfo capability file (UNIX) 131
kbd	specify keyboard layout (QNX) 133
kill	terminate or signal processes (POSIX) 135
less	interactive paginator (UNIX) 137
lex	lexical analyzer generator (UNIX) 151
ln	create links to (aliases for) files (POSIX) 153
login	log in to QNX (QNX) 155
logout	terminate a QNX Session (QNX) 157
lp	send files to a printer (POSIX) 159
lpc	line printer control program (QNX) 161
lpq	display print queue status (QNX) 163
lprm	remove jobs from the printer queue (QNX) 165
ls	list directory contents (POSIX) 167
make	maintain, update, and regenerate groups of programs (QNX) 171
mkdir	make directories (POSIX) 195
mkfifo	make FIFO special files (POSIX) 197
modem	modem handler to answer calls (QNX) 199

more	display files on a page-by-page basis (POSIX) 205
mount	mount partitions and filesystems (QNX) 209
mv	move files (POSIX) 211
nameloc	name locator (QNX) 215
netinfo	display network events (QNX) 217
newgrp	change to a new group (POSIX) 219
onnode	start a process on another node (QNX) 221
ontty	start a process on another tty (QNX) 223
passwd	change login password (QNX) 225
pr	print files (POSIX) 227
prefix	examine or change the pathname prefix tree (QNX) 231
ps	report process status (POSIX) 235
pwd	print working directory name (POSIX) 237
qcp	Quantum communications protocol (QNX) 239
qtalk	talk over communications line (QNX) 243
rm	remove files (POSIX) 251
rmdir	remove directories (POSIX) 253
rtc	set or get date from realtime clock (QNX) 255
sac	display system activity at each priority level (QNX) 257
sed	stream editor (POSIX) 259
sh	command interpreter (POSIX) 267
shutdown	shutdown and reboot the system (QNX) 269
sin	display system information (QNX) 271
slay	kill a process by name (QNX) 275
sleep	suspend execution for an interval (POSIX) 279
sort	sort, merge, or sequence-check text files (POSIX) 281
spatch	full-screen patch utility (QNX) 285
stty	set tty attributes (POSIX) 289

su	switch userid (UNIX) 297
tail	copy the last part of a file (POSIX) 299
textto	convert text files to QNX 2.1, QNX 4.0, or DOS format (QNX) 301
tic	terminfo compiler (UNIX) 303
ticksize	set the timeslice rate (QNX) 305
time	determine the execution time of a command (UNIX) 307
tinit	terminal initialization (QNX) 309
touch	change file access and modification times (POSIX) 313
tr	translate characters (POSIX) 317
umask	get or set the file mode creation mask (POSIX) 321
use	print a usage message (QNX) 325
usemsg	change usage message for a command (QNX) 327
vedit	VEDIT Text Editor (QNX) 331
wc	word, line, and byte count (POSIX) 333
which	locate a program file (QNX) 335
who	display who is on the system (QNX) 337
write	write a message to a user's terminal (UNIX) 339
ws	walk directory structure executing a command (QNX) 341
xargs	construct argument list(s) and execute program (POSIX) 347
yacc	a parser generator (UNIX) 351
zap	destroy a damaged file (QNX) 353

Utilities A-Z



Syntax: **awk** [-F *ere*] *program* [*argument...*]

awk [-F *ere*] [-f *progfile*] [*argument...*]

- Options:**
- F *ere* Define the input field separator (FS) to be the extended regular expression *ere*, before any input is read (see the section on “Regular expressions”).
 - f *progfile* Specifies the name of the **awk** program, *progfile*, which contains **awk** statements. You can also specify the **awk** program as a single argument in the command line by using quotes.
 - program* The text of an **awk** program that contains **awk** statements, assuming no -f *progfile* option has been specified. The **awk** program can either be in a file *progfile* or specified in the command line, taking into account the quoting rules of the shell.
 - argument* You can intermix either of these two types of arguments:
 - file* The pathname of a file that contains the input to be read; the input is matched against the set of patterns in the program. If you don’t specify any files containing input, or if you specify the dash character, (-), the standard input is used.
 - assignment* Expressions in the form *name=value* can be passed to **awk** where each instance of *name* represents the name of an **awk** variable. Each such variable assignment occurs just prior to the processing of the following *file*, if any. The assignment is done before the *file* argument is executed, and after the **BEGIN** action—if any—of that file.

awk

pattern scanning and processing language (POSIX)

Examples: Print the file **myfile**, which contains page references, filling in page numbers starting at 5:

```
awk '/Page/{ $2=n++; }  
      { print }' n=5 myfile
```

Description: The **awk** utility executes programs written in the **awk** programming language; this language specializes in manipulating textual data. An **awk** program is a sequence of patterns and corresponding actions. When input matching a specified pattern is read, the action associated with that pattern is carried out.

The **awk** utility interprets each input line as a sequence of fields, where, by default, a field is a string of non-**<blank>** characters. You can change this default white space delimiter by using the built-in variable, **FS** (see “Variables and special variables”), or the **-F** *ere* option. The **awk** utility denotes the first field in a line **\$1**, the second **\$2**, and so forth. A **\$0** refers to the entire line; setting any other field causes **\$0** to be reevaluated.

Each input line matched by the patterns and each input line for the **getline** function (see the section on “Functions”) is limited to 1024 bytes.

Programs in **awk** are composed of statements of the form:

```
pattern { action }
```

You can omit either the pattern or the action (that includes the enclosing braces). In the following sections of this description, **<blank>** characters between operators and reserved words are ignored, unless otherwise specified. All **<blank>** characters are significant inside literal strings and after a function name. A missing pattern matches any line of input, and a missing action is equivalent to an action that writes the matched line of input to standard output.

An **awk** program follows this general procedure:

- 1) Starts by executing the actions associated with all **BEGIN** patterns (**BEGIN** patterns are described in the section “Special patterns—**BEGIN** and **END**”).

- 2) Processes each file argument in turn—or standard input if no files are specified—by cyclically reading and saving data from the file until a record separator is seen, evaluating each pattern in the script and executing that action for patterns that evaluate to true (the default record separator is the **<newline>** character).
- 3) Executes the actions associated with all **END** patterns.

Expressions:

Expressions describe computations used in patterns and actions. Expressions in **awk** are constructed from such operators as conditionals, logicals, arithmetics, assignments, subscripts, and fields. Expressions take on string or numeric values as are appropriate to the context. The following table, “**awk** Expressions in Decreasing Precedence,” displays valid expressions in groups, starting from highest priority.

In the following table, the abbreviation *expr* represents any expression. The abbreviation *lvalue* represents any entity that can be assigned to (i.e. on the left side of an assignment operator).

awk Expressions in Decreasing Precedence

Syntax	Description
(<i>expr</i>)	grouping
\$ <i>expr</i>	references field number <i>expr</i>
++ <i>lvalue</i>	pre-increment <i>lvalue</i> by 1
-- <i>lvalue</i>	pre-decrement <i>lvalue</i> by 1
<i>lvalue</i> ++	post-increment <i>lvalue</i> by 1
<i>lvalue</i> --	post-decrement <i>lvalue</i> by 1
<i>expr</i> ^ <i>expr</i>	exponentiation
! <i>expr</i>	logical not
+ <i>expr</i>	unary plus
- <i>expr</i>	unary minus
<i>expr</i> * <i>expr</i>	multiplication
<i>expr</i> / <i>expr</i>	division

awk

pattern scanning and processing language (POSIX)

Syntax	Description
<i>expr % expr</i>	integer modulus
<i>expr + expr</i>	addition
<i>expr - expr</i>	subtraction
<i>expr expr</i>	string concatenation of two <i>exprs</i>
<i>expr < expr</i>	less than
<i>expr <=</i>	less than or equal to
<i>expr != expr</i>	not equal to
<i>expr == expr</i>	equal to
<i>expr > expr</i>	greater than
<i>expr >= expr</i>	greater than or equal to
<i>expr1 ~expr2</i>	1 if <i>expr1</i> matches the ERE described by <i>expr2</i>
<i>expr1 !~expr2</i>	1 if <i>expr2</i> doesn't match the ERE described by <i>expr2</i>
<i>expr in array</i>	1 if <i>array[expr]</i> exists
<i>(index) in array</i>	handles multi-dimensional arrays
<i>expr && expr</i>	logical AND
<i>expr expr</i>	logical OR
<i>expr?expr: expr</i>	conditional <i>expression</i> —evaluates first <i>expr</i> and if nonzero evaluates to the second <i>expr</i> ; otherwise to the third <i>expr</i>
<i>lvalue ^= expr</i>	raise <i>lvalue</i> to exponent <i>expr</i>
<i>lvalue %= expr</i>	assign <i>lvalue%expr</i> to <i>lvalue</i>
<i>lvalue *= expr</i>	multiply <i>lvalue</i> by <i>expr</i>
<i>lvalue /= expr</i>	divide <i>lvalue</i> by <i>expr</i>
<i>lvalue += expr</i>	add <i>expr</i> to <i>lvalue</i>
<i>lvalue -= expr</i>	subtract <i>expr</i> from <i>lvalue</i>
<i>lvalue = expr</i>	assign <i>expr</i> to <i>lvalue</i>

All of the arithmetic operators are based on the C Standard. The conditional expression returns either strings or numbers, depending on the input expressions. Only one of the alternative expressions is evaluated.

In addition to the descriptions in the previous table, an expression can also be a floating point number, a literal string enclosed by double quotation marks ("), or a variable name.

You can treat a variable or a field as a number or a string at any time, depending on its current usage. There are no explicit conversions between numbers and strings. To force an expression to be treated as a number, you add zero to it. To force an expression to be treated as a string, concatenate the null string ("") to it. Variables and fields are set by the assignment statement

lvalue=*expression*

and the type of *expression* determines the resulting variable type.

The assignment includes these arithmetic assignments:

`+=    -=    *=    /=    %=    ^=    ++    --`

each of which produces a numeric result. The left hand side of an assignment and the target of increment and decrement operators can be one of the following:

- a variable
- an array with index
- a field selector

This is indicated by the following BNF grammar:

```
<lvalue> :
            <variable>
            | $expr
            | <variable> [ <index> ]
            ;
```

A valid array index consists of one or more comma-separated expressions, with one expression for each dimension of the array. Because **awk** arrays behave as associate memories, an array index can be any string. Since **awk** arrays are really one-dimensional, a multi-dimensional array index is converted to a one-dimensional index by concatenating all expressions, each separated from the other

by the value of the **SUPSEP** variable (see “Variables and special variables”).

Thus, the following two index operations are equivalent:

```
var[expr1, expr2, ..., exprn]
var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]
```

A multi-dimensioned index used with the **in** operator must be parenthesized. The **in** operator, which tests for the existence of a particular array element, doesn’t cause that element to exist. But any other reference to a nonexistent array element automatically creates it.

Comparisons are made numerically if both operands are numeric; otherwise, operands are converted to strings as required and a string comparison is made.

In the previous table, “**awk** Expressions in Decreasing Performance,” operators of higher precedence are grouped before those of lower precedence. In expression evaluation, higher precedence operators are evaluated before lower precedence operators. All operators associate to the left except for the assignment operators, the conditional operator (**?:**), and the exponentiation operator (**^**). Because the concatenation operation is represented by adjacent expressions rather than an explicit operator, you often need to use parentheses to enforce the proper evaluation precedence.

Variables and special variables

You can use variables in an **awk** program by assigning to them. They don’t need to be declared—uninitialized variables have the value of the empty string, which has a numeric value of zero. All variables, including fields, are treated as string variables unless they’re used in a clearly numeric context.

Field variables are designated by a **\$**, followed by a number or a numerical expression.

You can create new field variables by assigning a value to them. References to non-existent fields—i.e. fields after **\$(NF)**—produce the null string. But, assigning to a non-existent field (e.g. **\$(NF+2)=5**) increases the value of **NF**, creates any intervening fields

with the null string as their values, and causes the value of **\$0** to be recomputed, with the fields being separated by the value of **OFS**.

Other special variables set by **awk** are shown in the following table:

awk Variables

Variable	meaning
ARGC	the number of elements in the ARGV array
ARGV	array of command-line arguments—excluding options and the program argument—numbered from zero to ARGC -1
FILENAME	pathname of the current input file
FNR	ordinal number of the current record in the current file
FS	input field separator regular expression; <space> by default
NF	number of fields in the current record
NR	ordinal number of the current record from the start of input
OFMT	print statement output format for numbers; "%.6g" by default
OFS	print statement output field separation; <space> by default.
ORS	print statement output record separator; <newline> by default
RLENGTH	length of string matched by the <i>match</i> function
RS	the first character of the string value of RS is the input record separator; <newline> by default. If RS is null, records are separated by blank lines, and <newline> is always a field separator, regardless of the value of FS
RSTART	starting position of string matched by <i>match</i> function, numbering from 1. This is always equivalent to the return value of the <i>match</i> function

SUBSEP subscript separator string for multi-dimensional arrays; the default value is `\034`

You can modify or add to the arguments in **ARGV**; you can alter **ARGC**. As each input file ends, **awk** treats the text non-null element of **ARGV**, up through the current value of **ARGC**-1, as the name of the next input file. Thus, setting an element of **ARGV** to null means that it isn't treated as an input file. A dash (-) filename indicates the standard input. If an argument contains an equals sign (=), this argument is treated as an assignment rather than as a *file* argument.

Patterns The structure of a *pattern* is specified by the following BNF grammar:

```
<pattern> :  
    BEGIN  
    | END  
    | <simple pattern>  
    | <simple pattern>, <simple pattern>  
    ;  
  
<simple pattern> :  
    <simple expression>  
    | /<ere>/  
    ;
```

In other words, a *pattern* is any valid *expression*, or an extended regular expression. In addition, a *pattern* can be a range specified by two of these patterns separated by a comma, or can be one of the two special patterns **BEGIN** or **END**.

Special
patterns—**BEGIN**
and **END**

The **awk** utility recognizes two special patterns, **BEGIN** and **END**. **BEGIN** is matched once and its associated action is executed before the first line of input is read and before command-line assignment is done. **END** is matched once and its associated action is executed after the last line of input has been read. These two patterns have associated actions.

BEGIN and **END** don't combine with other patterns. Multiple **BEGIN** and **END** patterns are allowed. The actions associated with the **BEGIN** patterns are executed in the order specified in the program, as are the

END actions. An **END** pattern can precede a **BEGIN** pattern in a program.

If a program consists of:	then:
only BEGIN blocks	awk exits without reading its input when the last statement in the BEGIN block is executed.
only END blocks or only BEGIN and END blocks	the input is read before the statements in the END block(s) are executed.

Regular expressions

The **awk** utility makes use of the extended expression notation, except that it allows the use of C language conventions for escaping special characters within the extended regular expressions, as you can see in the following table:

awk Regular Expression Escape Characters

Escape	meaning
<code>\b</code>	<backspace>
<code>\f</code>	<form-feed>
<code>\n</code>	<newline>
<code>\r</code>	<carriage return>
<code>\t</code>	<tab>
<code>\ddd</code>	1-3 digit octal value <i>ddd</i>

If *ere* is an extended regular expression, the pattern:

```
/ere/
```

matches any line of input that contains a substring specified by the regular expression. You can limit a regular expression comparison to a specific field or string by using one of the two regular expression matching operators, `~` and `!~`. For example:

```
$4 ~ /ere/
```


matches any line in which the fourth field matches the regular expression `/ere/`.

This pattern:

```
$4 !~ /ere/
```

matches any line in which the fourth field doesn't match the regular expression `/ere/`.

You can use an extended regular expression to separate fields by using the `-F ere` option, or by assigning the expression to the built-in variable `FS`. The default field separator is a single space character. The following describes the behavior of `FS`:

- 1) If `FS` is a single character:
 - a) If `FS` is `<space>`, skip leading and trailing `<blank>`s; fields are delimited by sets of one or more `<blank>`s.
 - b) If `FS` is any other character `c`, fields are delimited by each single occurrence of `c`.
- 2) Otherwise, if `FS` is more than one character, `FS` is considered to be an extended regular expression. Each occurrence of the string matching the extended regular expression delimits fields.

Pattern ranges

A pattern range consists of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the following occurrence of the second pattern, inclusive. At this point, the pattern range can be repeated starting at input lines subsequent to the end of the matched range.

Expression patterns

An expression pattern is considered to match—or be true—when the expression evaluates to a nonzero numeric value. Otherwise, the pattern is considered false.

Actions An action is a sequence of statements. A statement can be one of the statements listed as follows. In this list, optional elements are shown in square brackets ([]) and keywords are shown in a **constant-width** typeface.

```

if ( expression ) statement [else statement]
while ( expression ) statement
do statement while ( expression )
for ( [expression]; [expression]; [expression] ) statement
for ( variable in array ) statement
delete array[index]
break
continue
{ [statement]... }
variable = expression
next
exit [expression]
return [expression]
print [(] [expression-list] [)] [redirection-expression]
printf [(] format [, expression-list] [)] [redirection-expression]

```

Any single statement can be replaced by a statement list enclosed in braces (i.e. { }). The statements in a statement list are separated by **<newline>** characters or semicolons. The symbol # anywhere in a program line—in strings or EREs—begins a comment that is terminated by the end of the line.

Statements are terminated by semicolons or **<newline>** characters. You can split a long statement across several lines by ending each partial line with a backslash; **<newline>** characters without backslashes can follow:

- a comma
- an open brace
- a logical AND operator (&&)
- a logical OR operator (||)
- the **do** keyword

awk

pattern scanning and processing language (POSIX)

- the **else** keyword
- the closing parenthesis of an **if**, **for**, or **while** statement

For example:

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

String constants are surrounded by double quotes ("..."). A string expression is created by concatenating constants, variables, field names, array elements, functions, and other expressions.

The *expression* acting as the conditional in an **if** statement is evaluated, and if it is nonzero and non-null, the next statement is executed. Otherwise, if **else** is present, the statement following the **else** is executed.

The **while**, **do...while**, **for**, **break**, and **continue** statements are based on the C Standard, except in the case of **for (variable in array)**, which iterates assigning each *index* of *array* to *variable* in order. The **for** statement has a form that processes each element in an array. The order of processing is unspecified.

The **awk** language supplies arrays that are used for storing numbers or strings. Arrays don't have to be declared and their sizes change dynamically. The subscripts, or element identifiers, are strings, providing a type of associative array capability. Subscripts can't themselves be arrays.

The **delete** statement removes an individual array element. Thus, the following code deletes an entire array:

```
for (index in array)  
    delete array[index]
```

The **next** statement causes all further processing of the current input line to be abandoned.

The **exit** statement invokes all **END** actions in the order in which they occur in the program source. A **next** statement inside an **END** also terminates the program and can optionally set the utility's exit status.

Output statements

Both **print** and **printf** statements send their output to standard output by default. The output is written to the location specified by *redirection-expression*, if one is supplied, as follows:

```
>expression  
>>expression  
|expression
```

In all cases, the *expression* is evaluated to produce a string that's used as a full pathname to write into (for **>** or **>>**) or as a command to be executed (for **|**). Using the first two forms, if the file of that name isn't currently open, it is created if necessary, opened, and using the first form, truncated. The output is then appended to the file. Subsequent calls in which *expression* evaluates to the same name simply append output to the file; the file remains open until closed.

The third form writes output onto a stream compatible with *popen()*. If no stream is currently open, the stream is created with the same command name; the stream created is compatible with *popen()* invoked with a mode of **w**. Subsequent calls write output to the existing stream if, in those calls, *expression* evaluates to the same command name as a stream that's currently open. The stream is closed as if *pclose()* were called with an expression that evaluates to the same command name.

The **print** statement writes the value of each expression argument to the indicated output stream, separated by the current output field separator (see **OFS** in the table “**awk** Variables”), and terminated by the output record separator (see **ORS** in the table). String *expressions* are written out as is; numeric *expressions* are written out as if produced by **printf** using a format that is the string value of the variable **OFMT**. The *expression-list* is a comma-separated list of *expressions*. An empty *expression-list* stands for the whole input line (**\$0**).

With **printf**, the expressions are printed according to the specified format. A format argument is required—all other arguments in *expression-list* are optional. The string value of the expression *format* is interpreted in a manner similar to the C function **printf**, as follows. In the *format* string, format specifications begin with the

single character %, and can optionally include the following three modifiers:

- left justify the expression in its field
- width* pad left to this width as needed; a leading 0 pads with zeros
- .prec* maximum string width, or digits to right of decimal point

Format specifications are terminated by any other character. For each format specification that consumes an argument, the next argument from *expression-list* is evaluated and converted to the appropriate type (string, integer, or floating point). Both **print** and **printf** can output at least 1024 bytes.

awk printf Format Specification Characters

Character	Interpretation
c	if the argument is numeric, print a character; if the argument is a string, print only the first character
d	decimal integer
e	exponential notation: [-]d.dddddE[+-]dd
f	floating point: [-]ddd.ddddd
g	shorter of e or f notations; suppress nonsignificant zeros
o	unsigned octal number
s	string
x	unsigned hexadecimal number
%	print a %; no argument consumed

Functions: The **awk** language has a variety of built-in functions—arithmetic, string, input/output, and general.

Arithmetic Functions The arithmetic functions, except for **int**, are based on the C Standard.

atan2 (<i>y</i> , <i>x</i>)	returns arctangent of <i>y/x</i>
cos (<i>x</i>)	returns cosine of <i>x</i> , where <i>x</i> is in radians
sin (<i>x</i>)	returns sine of <i>x</i> , where <i>x</i> is in radians
exp (<i>x</i>)	returns the exponential function of <i>x</i>
log (<i>x</i>)	returns the natural logarithm of <i>x</i>
sqrt (<i>x</i>)	returns the square root of <i>x</i>
int (<i>x</i>)	truncates its argument to an integer; truncated toward 0 when <i>x</i> >0.
rand ()	returns a random number <i>n</i> , such that $0 \leq n < 1$.
srand ([<i>expr</i>])	sets the seed value for rand to <i>expr</i> or uses the time of day if <i>expr</i> is omitted. The previous seed value is returned.

String functions	gsub (<i>ere</i> , <i>repl</i> [, <i>in</i>])	Behaves like sub (see below), except that it replaces all occurrences of the regular expression in \$0 or in the <i>in</i> argument, when specified.
	index (<i>s</i> , <i>t</i>)	Returns the position, in characters, numbering from 1, in string <i>s</i> where string <i>t</i> first occurs, or zero if it doesn't occur at all.
	length [(<i>s</i>)]	Returns the length, in characters, of its argument taken as a string, or of the whole, \$0 , if there is no argument.
	match (<i>s</i> , <i>ere</i>)	Returns the position, in characters, numbering from 1, in string <i>s</i> where the extended regular expression <i>ere</i> occurs, or zero if it doesn't occur at all. RSTART is set to the starting position (which is the same as the returned value), zero if no

awk

pattern scanning and processing language (POSIX)

match is found; RLENGTH is set to the length of the matched string, -1 if no match is found.

split(*s*, *a* [, *fs*]) Splits the string *s* into array element *a*[1], *a*[2],..., *a*[*n*], and returns *n*. The separation is done with the extended regular expression *fs* or with the field separator FS if *fs* isn't given.

sprintf(*fmt*, *expr*, *expr*, ...)

Formats the expressions according to the **printf** format given by *fmt* and returns the resulting string.

sub(*ere*, *repl* [, *in*]) Substitutes the string *repl* in place of the first instance of the extended regular expression *ere* in string *in* and returns the number of substitutions. If *in* is omitted, **awk** substitutes in the current record (\$0).

substr(*s*, *m* [, *n*]) Returns the *n*-character substring of *s* that begins at position *m*, numbering from 1. If *n* is missing, the length of the substring is limited by the length of the string *s*.

All of the preceding functions that take *ere* as a parameter expect a pattern or a string valued expression that is a regular expression.

Input/Output and general functions

close(*expression*) Closes the file or pipe named *expression*.

expression | **getline** [*var*]

Pipes the output of the command string given by *expression* into **getline**; each successive call to **getline** returns the next line of output from *expression*. This construct has the behavior of *popen*() called with a *mode* of **r**. If *var* isn't specified, \$0 and NF are set; if *var* is specified, *var* is set.

getline Sets **\$0** to the next input record from the current input file. This form of **getline** sets **NF**, **NR**, and **FNR** variables.

getline < *expression*

Sets **\$0** to the next record from the pathname *expression*. This form of **getline** sets the **NF** variable.

getline *var* Sets variable *var* to the next input record from the current input file. this form of **getline** sets **FNR** and **NR** variables.

getline *var* < *expression*

Sets *var* from the next record of *expression*, treated as a pathname. This form of **getline** doesn't set the **NF**, **NR**, and **NFR** variables.

system (*expression*)

Executes the command given by *expression* in a manner consistent with the **system** function and returns the exit status.

All forms of **getline** return 1 for successful input, zero for end of file, and -1 for an error.

User-defined functions

The **awk** language also provides user-defined functions. You can define such functions—in the pattern position of a pattern-action statement—as:

```
function name (args, ...) { statements }
```

A function can be referred to anywhere in an **awk** program. In particular, a function call can precede its definition. The scope of a function is global.

Function arguments are passed by value if scalar and by reference if an array name. Argument names are local to the function; all other variable names are global. The number of parameters in the function definition doesn't have to match the number of parameters in the

awk

pattern scanning and processing language (POSIX)

function call. Excess formal parameters can be used as local variables. If fewer arguments are supplied in a function call than are in the function definition, the extra receiving parameters are left uninitialized.

When you're invoking a function, remember that no white space is allowed between the function name and the opening parenthesis. Function calls can be nested and can be recursive. You can use the **return** statement to return a value.

In the function definition, **<newline>**s are optional before the opening brace and after the closing brace. Function definitions can appear anywhere in the program where a pattern-action statement is allowed. In a function call, no white space is allowed between the function name and the opening parenthesis that begins the function parameter list.

Sample **awk**
programs:

Note that the following are sample **awk** programs, not complete command lines.

Write to the standard output all input lines for which field 3 is greater than 5:

```
$3 > 5
```

Print every tenth line:

```
(NR % 10) == 0
```

Print any line with a substring that matches the regular expression:

```
/(G|D) (2[0-9][[:alpha:]]*)/
```

Print the second to last field and the last field in each line:

```
{OFS=" "; print $(NF-1), $NF}
```

Print the line number and number of fields in each line. The three strings representing the line number, the colon, and the number of fields are concatenated and the resulting string is written to standard output:

```
{print NR ":" NF}
```

Print lines that are longer than 72 characters:

```
length $0 > 72
```

Print the first two fields in opposite order separated by the OFS:

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

Same as above, with input fields separated by a comma or <space> and <tab> characters, or a combination of all these:

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

Add up the first column, and print the sum and the average:

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

Print the fields in reverse order, one field per line (i.e. many lines out for each line in):

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between occurrences of the strings **start** and **stop**:

```
/start/, /stop/
```

Print all lines whose first field is different from the first field of the previous line:

```
$1 != prev { print; prev = $1 }
```

Simulate **echo**:

```
BEGIN {  
    for (i = 1; i < ARGV; ++i)  
        printf "%s%s", ARGV[i], i==ARGV-1?"\n":" "  
    }  
}
```

If there's a file named **myfile** that contains page headers of the form:

```
Page #
```

awk

pattern scanning and processing language (POSIX)

and a file named **program** that contains:

```
/Page/{ $2 = n++; }  
{ print }
```

then the command line:

```
awk -f program n=5 myfile
```

would print the file **myfile**, filling in page numbers starting at 5.

Exit status: 0 All input files were processed successfully.
>0 An error occurred.

Environment variables: **PATH** Defines the search path when looking for commands executed by **system(*expr*)**, or input and output pipes.

Files: By default, input files are text files that are read in order. You can modify either variable **ARGV** or variable **ARGC** to place this default file processing under program control.

The nature of the output files depends on your **awk** program.

See also: A.V. Aho, Brian W. Kernighan, and Peter J. Weinberger, *The AWK Programming Language* (Addison-Wesley, 1988)

Syntax: **basename** *string* [*suffix*]

Options: *string* A string of text.
 suffix A string of text.

Examples:	Command:	Output:	Exit status:
	basename /user/src/prog.c	prog.c	0
	basename /user/src/prog.c .c	prog	0
	basename /user/src/prog.c .a	prog.c	0
	basename /user/src/		0
	basename ../[fred]	[fred]	0*

* [fred] isn't a valid filename, but this isn't checked

Description: The **basename** utility is primarily useful for extracting the filename portion of a pathname, but since it performs only string operations, it can be used with any string.

The **basename** utility prints to standard output a substring of its *string* argument, plus a <newline> character. The **basename** utility forms this substring by:

- deleting from the pathname all characters up to and including the last slash (/) character in the *string* argument
- deleting the *string* argument's suffix (provided you've specified a *suffix* argument that is identical to the *string* argument's suffix)

The result may be a null string if any of the following conditions are met:

- the *string* argument ends in the slash character
- the *string* argument is the same as the *suffix* argument
- the portion of the *string* argument after the last slash is the same as the *suffix*

In these cases, **basename** outputs a single <newline> character.

You'll use the **basename** utility most often within shell procedures, where it's normally invoked inside backquotes (` `), or preceded by \$(). A typical example is the following procedure, which works in the current directory, finds C source files (.c suffix), and deletes

basename

return nondirectory portion of pathname (POSIX)

matching object modules (.o or .obj suffix) and executables (no suffix):

```
for i in *.c                # loop through .c files
do
    x = `basename $i .c`    # name - suffix = executable
    y = ${x}.o              # add .o suffix
    z = ${x}.obj            # add .obj suffix
    for j in $x $y $z
    do
        if [ -f $j ]
        then
            rm $j
        fi
    done
done
```

To extend a procedure like this one beyond the current directory, you use both **basename** and **dirname** (**dirname** extracts the directory part of a pathname), as follows:

```
# error-checking for valid directory name in $1 omitted

for i in $(find $1 -name '*.c' -print)    # loop through .c files
do
    f = `basename $i .c`                # filename, minus .c
    d = `dirname $i`                    # rest of pathname
    x = ${d}/${f}                       # combine them
    y = ${x}.o                          # add .o suffix
    z = ${x}.obj                        # add .obj suffix
    for j in $x $y $z                   # loop through objects & executables
    do
        if [ -f $j ]                   # if it exists
        then
            rm $j                       # remove it
        fi
    done
done
```

Exit status: 0 Successful completion.
 >0 An error occurred.

See also: **dirname**

Syntax: `cat [-c] [-u] [file...]`

Options:

- `-c` Compress each sequence of multiple **<newline>** characters into a single **<newline>** character.
- `-u` Write unbuffered output. Data from the input file(s) is written to the standard output without delay as each file is read.

file The pathname of an input file. If no files are specified, the standard input is used. If a *file* is the dash character (-), **cat** reads from the standard input at that point in the sequence.

Examples: Write the contents of the file **myfile** to the standard output:

```
cat myfile
```

Concatenate the files **doc1** and **doc2** and write the result to the **doc.all** file:

```
cat doc1 doc2 > doc.all
```

Description: The **cat** utility reads files in the order you specify and writes the contents of these files to the standard output.

Exit status:

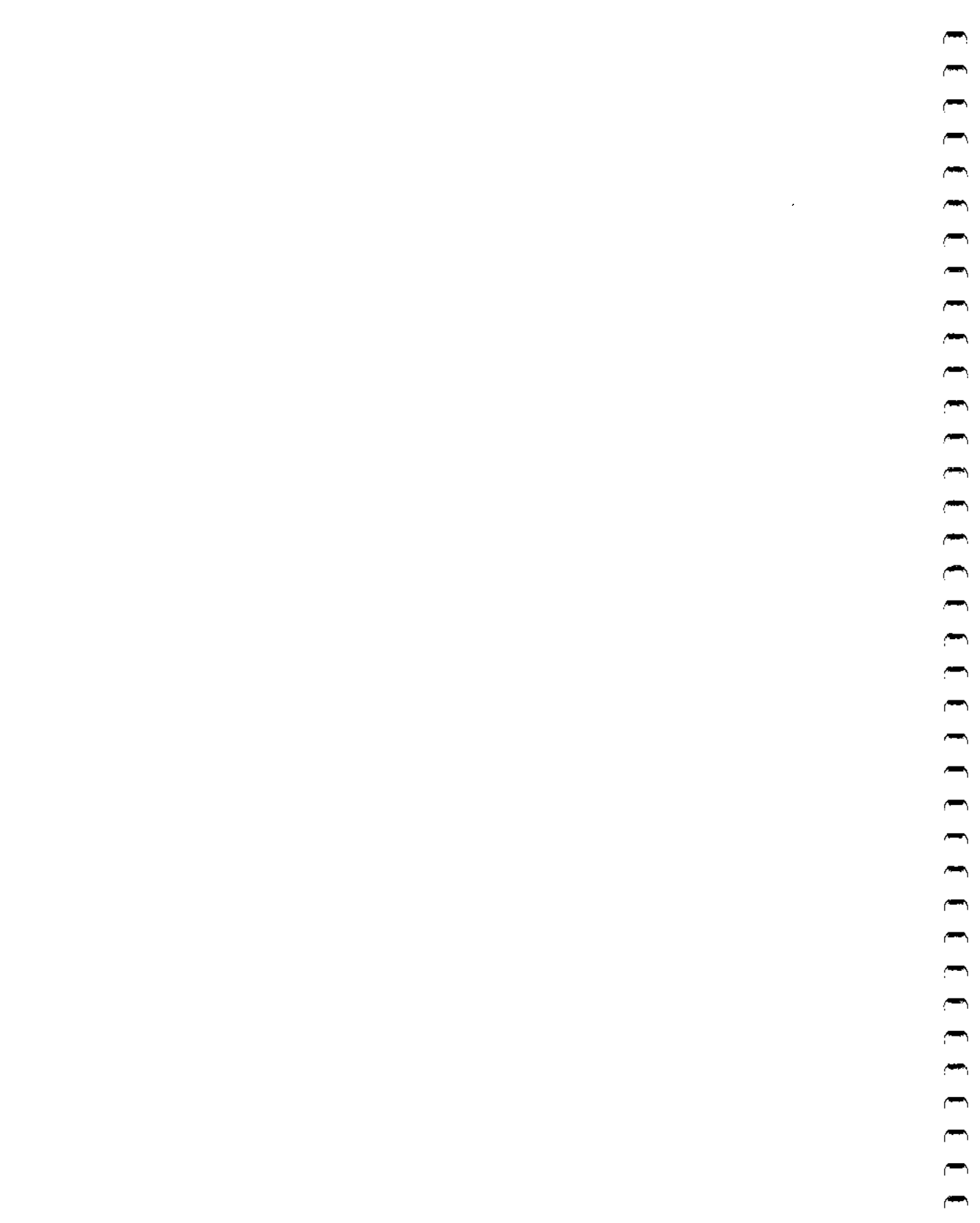
- 0 All input files were output successfully.
- >0 An error occurred.

See also: **head**, **tail**

Caveats: Because of the shell language mechanism used to perform output redirection, a command such as:

```
cat doc doc.end > doc
```

causes the original data in **doc** to be lost. The file **doc** would be opened for write by the shell, and therefore truncated to zero length, before **cat** was executed.



Syntax: `cc [options]... [operands]...`

OR

`c89 [options]... [operands]...`

- Options:**
- A *library*** Combine all object files provided as operands and build a new library. The *library* filename is given the **.lib** suffix.
 - b** Search the beta library that corresponds to the specified memory model (see option **-m**).

The beta libraries are searched before the standard libraries. The beta libraries include updates made since the last formal release of the software and can be updated by modem using the Quantum update system.
 - B** Search the QNX 2.1 migration library that corresponds to the specified memory model (see option **-m**). The QNX 2.1 migration library provides backwards compatibility for many discontinued functions. The migration library is searched before the beta and standard libraries.
 - c** Suppress the link phase and retain all created object files.
 - C{d|i|n}** Call routines in the system shared library directly (**-Cd**), indirectly (**-Ci**), or not at all (**-Cn**). If you choose (**-Cn**), the system library code will be linked directly into your program. The default is **-Cd**.
 - D *name=text*** Define *name* as if by a C language *#define* directive. If no *text* is specified, *name* is defined as 1.
 - E** Copy the language source files to the standard output, expanding all preprocessor directives. No compilation is performed. See also option **-P**.
 - fc** Use floating point calls into a linked-in library rather than generating inline 80x87 instructions.

- g{1|2}** Produce symbolic information in the object file and in the executable file. This option is necessary to run the symbolic debugger. If **-g1** is specified, the optimizer remains enabled for code generation. If **-g2** is specified, the optimizer is disabled, resulting in code that is easier to debug. Specifying **-g** is the same as specifying **-g2**.
- I *directory*** Change the default search algorithm for header files whose names don't begin with a slash (/) so that the directory named by the *directory* pathname is searched before the standard path, which is **/usr/include**. Headers are searched in the following order:
- 1) in the directory of the file with the *#include* line
 - 2) in the directories named in the **-I** option
 - 3) in **/usr/include**
 - 4) in **../h**
- Directories named with the **-I** option are searched in the order specified. If an *#include* is surrounded by angle brackets (<>), then the first search is *not* done.
- l *library*** Search the specified library file for any unresolved references during the link phase. Libraries are searched in the order they appear on the command line, and before the default libraries. If the library name doesn't start with a slash (/), a search is made in the following order:
- 1) in the current directory
 - 2) in the directories named in the **-L** option
 - 3) in the default directory
- L *directory*** Change the default algorithm (that searches for libraries) to include the specified directory. Directories are searched in the order they appear on the command line, and before the default directory. The default directory is **/usr/lib**. You may change

this default by setting the environment variable **LIBQNX** to one or more directories separated by colons.

-m{s|c|m|l|h}

Compile the program to conform to the specified memory model.

- s** small model
(64K code, 64K data)
- c** compact model
(64K code, >64K data)
- m** medium model
(>64K code, 64K data)
- l** large model
(>64K code, >64K data)
- h** huge model
(>64K code, >64K data
with objects >64K)

-M Produce a symbolic map file with the same name as the output file, but with a **.map** suffix.

-N *stacksize* Tell the linker to reserve room for a stack of *stacksize* bytes. The default stack size is 4000 bytes.

-o *outfile* Use the pathname *outfile* instead of the default **a.out** for the executable file. If the **-c** option is also specified, this option is ignored.

-O{a,d,i,l,s,t,x}

Enable compiler optimizations. If only **-O** is specified, the default is equivalent to **-Oi1**. You may specify more than one optimization by providing multiple letters together with no space or punctuation (e.g. **-Oadx**).

- a** Relax alias checking. When you specify this option, the code optimizer assumes that, within the same routine, no variable is indexed both

directly by name and indirectly by a pointer. This assumption may reduce the size of the code that is generated. The following example helps to illustrate this point:

```
extern int i;

void rtn( int *pi ) {

    int k;
    for( k = 0; k < 10; ++k ) {
        (*pi)++;
        i++;
    }
}
```

In this example, if `i` and `*pi` referenced the same integer object, `i` would be incremented by 2 each time through the `for` loop. The pointer reference `*pi` would be an alias for the variable `i`. As a result, in binding the variable `i` to a register, the compiler would have to make sure the “in-memory” copy of `i` was kept up-to-date. It’s unlikely this situation would arise, since you probably wouldn’t reference the same variable directly by name and indirectly by a pointer. The `-Oa` option instructs the code generator that such cases don’t arise in the module to be compiled. The code generator is able to produce more efficient code when it doesn’t have to worry about the alias “problem.”

- d Generate non-optimized code sequences. The resulting code is much easier to debug with the `wvideo` command (see the *WATCOM VIDEO User’s Guide*). By default, `cc` selects `-Od` if you specify option `-g`. But if option `-g` is followed by one of the other `-O` options, as in:

```
cc -g -Os report
```

the default `-Od` is overridden.

- i** Generate certain library functions in-line. For a function to be generated in-line, you must include the appropriate header file containing the prototype for the function. The functions that can be generated in-line are:

<i>abs()</i>	<i>_disable()</i>	<i>div()</i>
<i>_enable()</i>	<i>fabs()</i>	<i>_fmemchr()</i>
<i>_fmemcmp()</i>	<i>_fmemcpy()</i>	<i>_fmemset()</i>
<i>_fstrcat()</i>	<i>_fstrcmp()</i>	<i>_fstrcpy()</i>
<i>_fstrlen()</i>	<i>labs()</i>	<i>inp()</i>
<i>inpw()</i>	<i>memchr()</i>	<i>memcmp()</i>
<i>memcpy()</i>	<i>memset()</i>	<i>movedata()</i>
<i>outp()</i>	<i>outpw()</i>	<i>_rotl()</i>
<i>_rotr()</i>	<i>strcat()</i>	<i>strchr()</i>
<i>strcmp()</i>	<i>strcpy()</i>	<i>strlen()</i>

- l** (the lowercase letter “e”) Perform loop optimizations. This includes moving loop-invariant expressions outside the loops.
- s** Favor space over time when generating code. This results in smaller code but possibly slower execution.
- t** Favor time over space when generating code. This results in faster execution but possibly larger code.
- x** Select the **-Oi**, **-Ol**, and **Wc**, **-s** (no stack overflow checking) options. When you combine **-Ox** with the **-Oa** and **-Ot** options, the code generator attempts to give you the fastest executing code possible.

By default, **cc** selects a balance between “space” and “time.”

- P** Run only the C preprocessor and leave the output in the same name with the suffix replaced by **.i**. See also option **-E**.
- Q** Compile more quietly without printing the details of each compiler phase.

- s Strip symbolic information that may exist in object files that were compiled with the **-g** option. The symbolic information isn't removed from these object files; however, the resulting executable file is stripped of all symbolic information.
- S Produce assembly files ending in **.a** for all source or object files provided. If an object file is specified, the original source lines won't be included in the generated assembly.
- U *name* Remove any initial definition of *name* that the compiler itself might define. This doesn't affect any names defined in included header files.
- W*c,option* Pass the command specified by *option* off to the language processor, which is specified by the lowercase italic letter *c*. The language processor is typically the letter **c** for the C language. You may specify *c* as one of the following:
 - a** the assembler
 - c** the C compiler
 - d** the Disassembler (used in conjunction with **-S**)
 - l** the linker
 - p** the preprocessor (used in conjunction with **-E** or **-P**)

If *option* contains a space, you must enclose the entire option in double quotes (e.g. "**-Wc, -x file1**").

The command specified by *option* is defined by the language processor being used. Please refer to the appropriate documentation.
- w *warning_level*

Use the specified warning level. If this isn't specified, the built-in default for the language processor will be used.
- X Print, but don't execute, the steps required to compile or link.

Examples:	<code>cc test.c</code>	Compile <code>test.c</code> and create an executable program in the current directory with the name <code>a.out</code> .
	<code>cc -o test test.c</code>	Compile <code>test.c</code> and create an executable program in the current directory with the name <code>test</code> .
	<code>make test</code>	Compile <code>test.c</code> and create an executable program in the current directory with the name <code>test</code> . Use the default rules to invoke <code>cc</code> (see the <code>make</code> utility).
	<code>cc -c test.c</code>	Compile <code>test.c</code> and create an object file in the current directory with the name <code>test.o</code> .

Description: The `cc` command is the QNX compiler interface. It is based on the POSIX `c89` utility.

The `cc` utility takes a list of source and object modules on the command line and invokes the appropriate parser to compile each file. Object modules are passed straight through to the linker. The file suffix determines which parser is used, as follows:

- `.a` Assembler
- `.b` BASIC compiler
- `.c` C compiler
- `.f` FORTRAN compiler
- `.p` Pascal compiler

The parser must exist in order for you to use it. The `cc` command was designed to work with potential future parsers and may contain entries for parsers that aren't yet available.

The `cc` command doesn't allow multiple options to be specified after a dash character (`-`). For example, `-bc` isn't valid; you must specify `-b -c` instead. Operands (source and object files) and options may be mixed and specified in any order. Some options like `-I` and `-L` are order-dependant—the order in which they appear in the command line determines the order of the searches that are made. All

command-line arguments are processed before any compilation or linking begins.

When **cc** encounters a compilation error that stops an object file from being created, it writes a diagnostic to the standard error and continues to compile other source files, but it bypasses the link phase and returns a non-zero exit status. Any files that generated warnings or errors produce a file with the same name and with **.err** as its suffix. If the link phase is entered and is unsuccessful, a diagnostic is written and **cc** exits, once again, with a non-zero status.

The **-c** option suppresses the LINK phase. If you have many separate source files that you must update and modify individually, you will probably use the **-c** option frequently.

You may occasionally wish to examine the assembly code produced by the code generator. The **-S** option produces an assembly file ending in **.a**. You can also take an object file and convert it back to assembly with this option.

If you need to specify a parameter to any of the language processors, you may use the **-wc,option**. Check the documentation on each processor to determine its options.

Environment variables:

LIBQNX The list of library directories to search—directories are separated by colons. If **LIBQNX** isn't defined, then **cc** uses:

/usr/lib

If **LIBQNX** is defined, it is used as is.

Files:

The default output file is named **a.out**. This default may be changed with the **-o** option.



Note that the **make** utility, when used with the default settings, produces an output file with the same name as the input file. For example:

```
make file1
```

results in the executable output file: **file1**.

See also: **lex, make, yacc**



Syntax: `cd [directory]`

Options: *directory* The pathname of the directory that will become the new working directory. The interpretation of a relative pathname by `cd` depends on the **CDPATH** environment variable (see below).

Examples:

<code>cd</code>	change directory to the HOME directory
<code>cd ..</code>	change to the next highest directory in the directory structure
<code>cd /</code>	change directory to the root directory
<code>cd /path/dir</code>	set <code>/path/dir</code> as the current working directory
<code>cd ../..</code>	change to the directory two levels above the current working directory
<code>cd ../../files</code>	change to the directory two levels above the current working directory, then to the files directory below in the directory tree



You can use the `pwd` command to display the current working directory.

Description: The `cd` command is a shell built-in that changes the current or *working directory* of the invoking process to the directory named as its argument.

All filenames that don't start with a slash (/) are assumed to be relative to your current working directory.

The command:

`cd`

with no arguments changes the current working directory to your **HOME** directory set when you logged in.

cd

change working directory (POSIX)

This is equivalent to:

```
cd $HOME
```

if the **HOME** environment variable is set.

If the **HOME** environment variable isn't defined, or has the null string as its value, then the current working directory is unchanged.

The **cd** utility makes use of **CDPATH**, which is analogous to the use of the **PATH** variable in the shell. It lets you name a directory with a one-component name rather than a full pathname.

CDPATH is a list of pathnames, separated by colons. If the *directory* specified in the **cd** command doesn't begin with a slash (/), and the first component isn't dot or dot-dot, **cd** searches for *directory* relative to each path named in the **CDPATH** variable, in the order listed. The new working directory is set to the first matching path found. An empty string in place of a directory pathname represents the current directory. If **CDPATH** isn't set, it is treated as if it were an empty string.

For example, if **CDPATH** is set to:

```
/user/fred/:
```

the command:

```
cd newdir
```

looks for the **/user/fred/newdir** and for the **./newdir** directories and makes the first one of those it finds the working directory.

Note that if **CDPATH** is defined, then only those elements defined in **CDPATH** will be searched for relative directories.

Exit status:

- 0 The directory was successfully changed.
- >0 An error occurred; the directory remains unchanged.

**Environment
variables:****CDPATH**

When defined, **CDPATH** lists pathname components that are applied to any specified directory that doesn't begin with a slash (/).

HOME

The name of the home directory, used when no *directory* operand is specified.

See also:

pwd



Syntax: `chgrp [-R] group file...`

Options: `-R` Recursively change group ownership of files. For each *file* that names a directory, **chgrp** changes the group of the directory and of all files in the file hierarchy below it.

group A group name from the group database, or a numeric group ID.

file The pathname of a file whose group ID is to be modified.

Examples: `chgrp 27 data`

`chgrp technical data`

Description: The **chgrp** utility lets you change group ownership of one or more files by setting the group ID. For each file you name, **chgrp** sets the file's group ID to the group ID specified by the *group* operand.

Exit status: 0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

Consequences of errors: If you invoke **chgrp** with the `-R` option, and **chgrp** attempts but fails to change the group ID of a particular file in a specified file hierarchy, it continues to process the remaining files in the hierarchy. The **chgrp** utility can fail to change the group ID of a file if you don't have appropriate permissions.

See also: `chmod`, `chown`



Syntax: `chkfsys [-PRfpqrsv] drive`

Options:

- `-P` Don't prompt (fix without any questions).
- `-R` Enter Rambo mode! Override filesystem locks and allow fixes while files are open on the filesystem (this dangerous option disables the `-rP` options).
- `-f` Don't fix anything.
- `-p` Prompt before starting.
- `-q` Be quiet.
- `-r` Rebuild the bitmap without prompts or messages.
- `-s` Suppress statistics display.
- `-v` Be verbose.

drive The disk (e.g. `/dev/fd0` or `/dev/hd0t77` or `/` etc.).

Examples:

<code>chkfsys /dev/hd0t77</code>	Check filesystem on the QNX partition of a hard disk
<code>chkfsys -rs /</code>	Check filesystem mounted as the root (/)

Description: The **chkfsys** utility performs a consistency check of the filesystem on the requested drive. It will recursively walk the filesystem, checking every file on the disk. During the walk, checks are made on the directory entry of each file and the extents that make up the file. A bitmap is constructed in memory that is consistent with the block allocation of all files and directories on the disk. This bitmap is then compared to the existing one on the disk. If they differ, the user is given the option of replacing the existing bitmap on disk with the one constructed in memory.

chkfsys

check an entire filesystem for consistency (QNX)



The **chkfsys** utility should be used only when the filesystem is idle. There should be NO open files when **chkfsys** is running. In the normal mode, **chkfsys** will prevent its own operation when any files are open on the drive. If **chkfsys** finds that no files are open on the drive, it will proceed and will also prevent any files from being opened on the drive for the duration of its check.

If you're not doing any fixes (with the **-f** option), you may check a filesystem with open files, but *beware*: you may get inconsistent reports in this case. Also, if you really must fix the filesystem on an active drive (perhaps an application that requires the machine to be running 24 hours a day), you may specify the **-R** (Rambo) option. Note that the **-r** (automatic bitmap rebuild) and **-P** (prompt suppression) options are disabled when the **-R** option is used.

The **chkfsys** utility is normally used to recover blocks that were lost through the use of the **zap** utility. For example:

```
zap file1
```

In this case, **chkfsys** will complain that there are blocks used in the bitmap that are in fact *not* used by any file. These blocks may be recovered by writing the reconstructed bitmap back to disk. The **chkfsys** utility will attempt to read each of these blocks, but will *not* mark bad blocks as available. Any blocks found this way will be added to the **/ .bad_blks** file at the root of the filesystem being checked.

The **chkfsys** utility will tell you if any files are using blocks that are now known to be bad.

If **chkfsys** complains that a block is used by more than one file, this could indicate one of two problems. If bad blocks exist, then this means that the file uses a block that is bad and marked as used in the bitmap. If there are no known bad blocks, then a multiple allocation of a single block has occurred. In either case, the file should be saved on *another* disk (if possible), and the original file should be destroyed with **zap**. The **chkfsys** utility should then be run again to update the bitmap, after which the saved file may be restored onto this disk.

In general, whenever the bitmap is replaced, **chkfsys** should be run a second time to ensure that the filesystem is indeed consistent.

The **-f** (no fix) option will prevent **chkfsys** from attempting to make any fixes to the filesystem. The disk will not be opened for write, but only for read. This option lets a user examine the filesystem without requiring other users to stop using the disk or filesystem. Beware, however, that the **-f** option may report errors that don't really exist (if other users are opening, closing, or growing files during the time that **chkfsys** is running). Even so, this can be a valuable option for sites that are up and running 24 hours a day, when the system operator carefully evaluates the results. If you see errors that you believe result from current activity, run **chkfsys** again with the **-f** option to verify the errors. When you have located errors that require fixing, you should idle the filesystem and run **chkfsys** without the **-f** (no fix) option.

The **-p** (pause) option is used primarily with floppy disks. You can start **chkfsys** from a floppy diskette, wait for **chkfsys** to pause, remove the current disk (which contains the **chkfsys** command), and then insert another disk you wish to check.

The **-q** (quiet) option suppresses the display of each filename as that file is checked. This will speed the checking significantly, without loss of information, because **chkfsys** will show you the name of any files that have errors.

The **-r** (rebuild) option suppresses the warning message that normally appears at the end of a **chkfsys** run when the existing bitmap differs from the newly constructed bitmap in memory. When **-r** is specified, **chkfsys** will automatically rebuild the bitmap. Note that this option is not effective with the **-f** (no fix) option and cannot be turned on when the **-R** (Rambo) option is specified.

The **-s** (no stats) option prevents the display of the statistics message that normally appears at the end of a **chkfsys** run.

The **-v** (verbose) option will display information on the checking.

The **-P** (no prompt) option will cause **chkfsys** to automatically fix problems encountered without prompting the user before each fix.

chkfsys

check an entire filesystem for consistency (QNX)

Note that the **-P** option won't work with the **-f** option and can't be turned on when the **-R** option is specified.

The **-R** (Rambo) option is very dangerous. It lets you make fixes to the filesystem when files are open. For this reason, the **-R** option should not be used unless you are certain that:

- the filesystem can't be made idle
- no currently open files have been grown; if some files are known to have grown, you shouldn't fix problems found with these files, nor re-write the bitmap
- no files will be opened or created while **chkfsys** is running

Sometimes the **-R** option can come in handy. For example, suppose you have a database that must remain up and running all the time, but you suspect problems. Knowing that the open files will never grow in length, you could use the **-R** option.

If you use the **-R** option, you won't be able to also use either the **-r** or **-P** options—**chkfsys** will disable their use, even if specified. This will require you to consider each fix to be made as **chkfsys** finds a problem. If you want to be reminded at each problem (if any) that you're using the **-R** option, simply double it (i.e. **-RR**).

Note that making the filesystem idle and running **chkfsys** without the **-R** option is much safer and highly recommended.

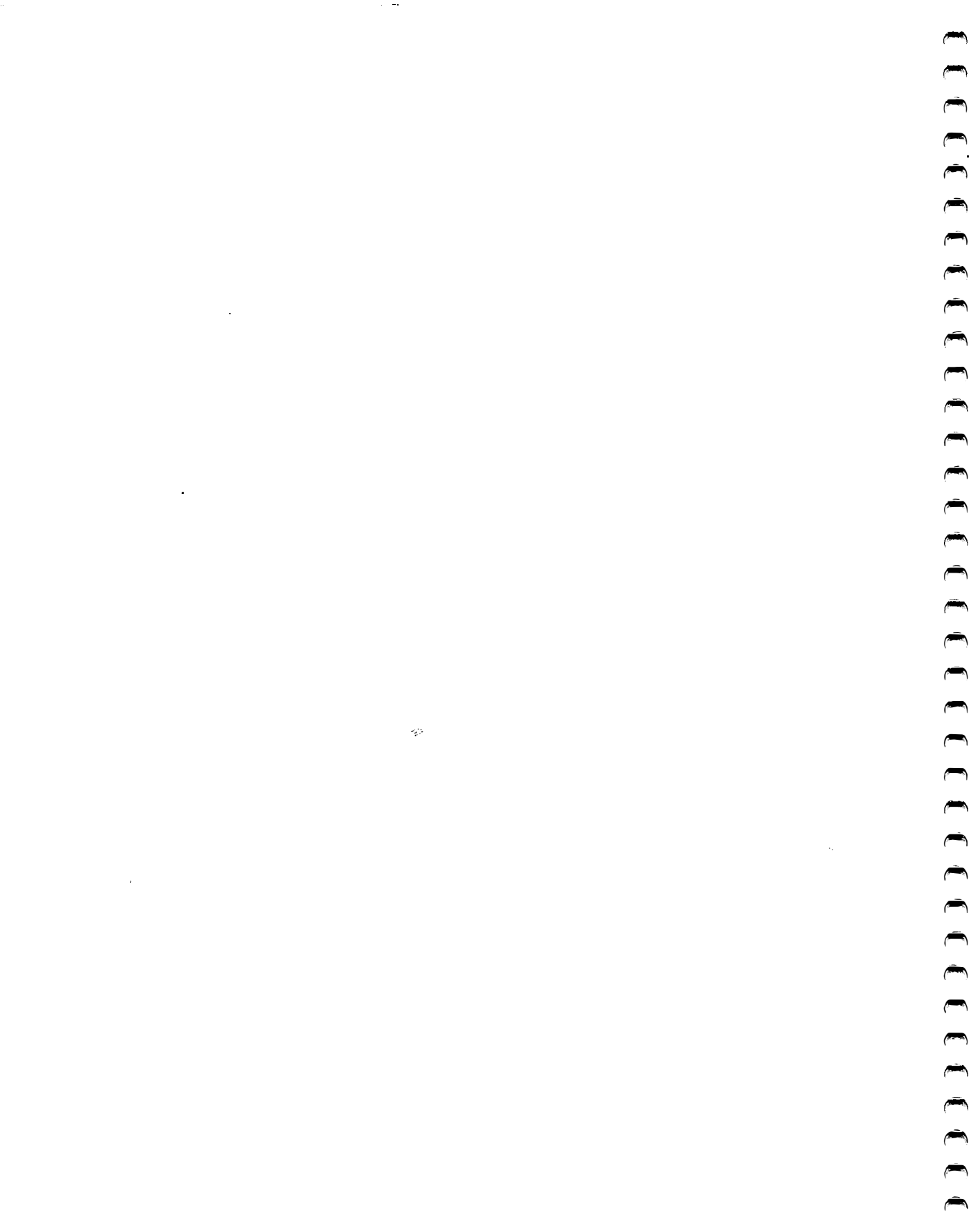
After a power failure

The **chkfsys** utility may also be run after a system crash or power failure, which may have left some files busy. The utility will make the files "unbusy" and will also make checks to ensure that no damage to the filesystem has occurred. QNX is designed to be very immune to this type of damage.

In the event of the loss of a filesystem due to the corruption of the root directory and the bitmap (first few blocks of the disk), you should refer to the **dinit** utility documentation for methods of initializing just those portions of your disk. If only the root block and bitmap are damaged, **chkfsys** will be able to recover the files in most cases. If the root directory or inode file is damaged (the next areas on the disk), recovery may be possible with the **dinit**,

spatch, and **chkfsys** utilities. You shouldn't expect to run into such problems; these are very rare events that we have simply tried to anticipate.

See also: **dcheck, dinit, spatch, zap**



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

Options:

- R** Recursively change file modes. For each *file* that names a directory, **chmod** changes the file mode bits of the directory and all files in the file hierarchy below it.
- mode* Represents the change to be made to the file mode of each *file* named (see the Description below).
- file* The pathname of a file whose file mode bits are to be modified.

Description: The **chmod** utility lets you change any or all of the file permission mode bits of one or more files. For each file you name, **chmod** changes the permission file mode bits according to the *mode* operand.

To change a file's permission mode bits, the user of **chmod** must be either the owner of the file or the superuser.

The *mode* option can be either a *symbolic_mode* expression or a non-negative octal integer.

Symbolic Modes The *symbolic_mode* has the following form:

`[[agou] [+==] [rswx]] [,symbolic_mode]`

The options of the symbolic form are:

a	user, group, and other access
g	group access
o	other access
u	user access
+	add specified permissions to the group, other, or user category of the specified files
-	remove specified permissions from the group, other, or user category of the specified files
=	set the specified permissions for the group, other, or user category of the specified files

chmod

change file modes (POSIX)

- r** read permission
- s** set user ID or group ID when executed
- w** write permission
- x** execute permission

Here are some examples of symbolic modes:

Make **myfile** executable by all:

```
chmod a+x myfile
```

Remove read permission for group and others:

```
chmod og-r myfile
```

Perform both the above operations, in the order given, on three files: **myfile**, **file2**, and **zzz**

```
chmod a+x,og-r myfile file2 zzz
```

Octal Modes

Each permission may be specified with an octal number: read = 4; write = 2; execute = 1; no permission = 0. The octal equivalents are derived by adding the numbers associated with the four basic permissions. The following table illustrates their use:

Octal number	Permission
0	--- none
1	--x execute
2	-w- write
3	-wx write/execute
4	r-- read
5	r-x read/execute
6	rw- read/write
7	rxw read/write/execute

In the octal form, the three octal digits represent user, group, and other permissions in that order.

Example of octal modes:

```
chmod 754 myfile
```

The user gets **7** (rwx), group **5** (r-x) and others **4** (r--).

Exit status:

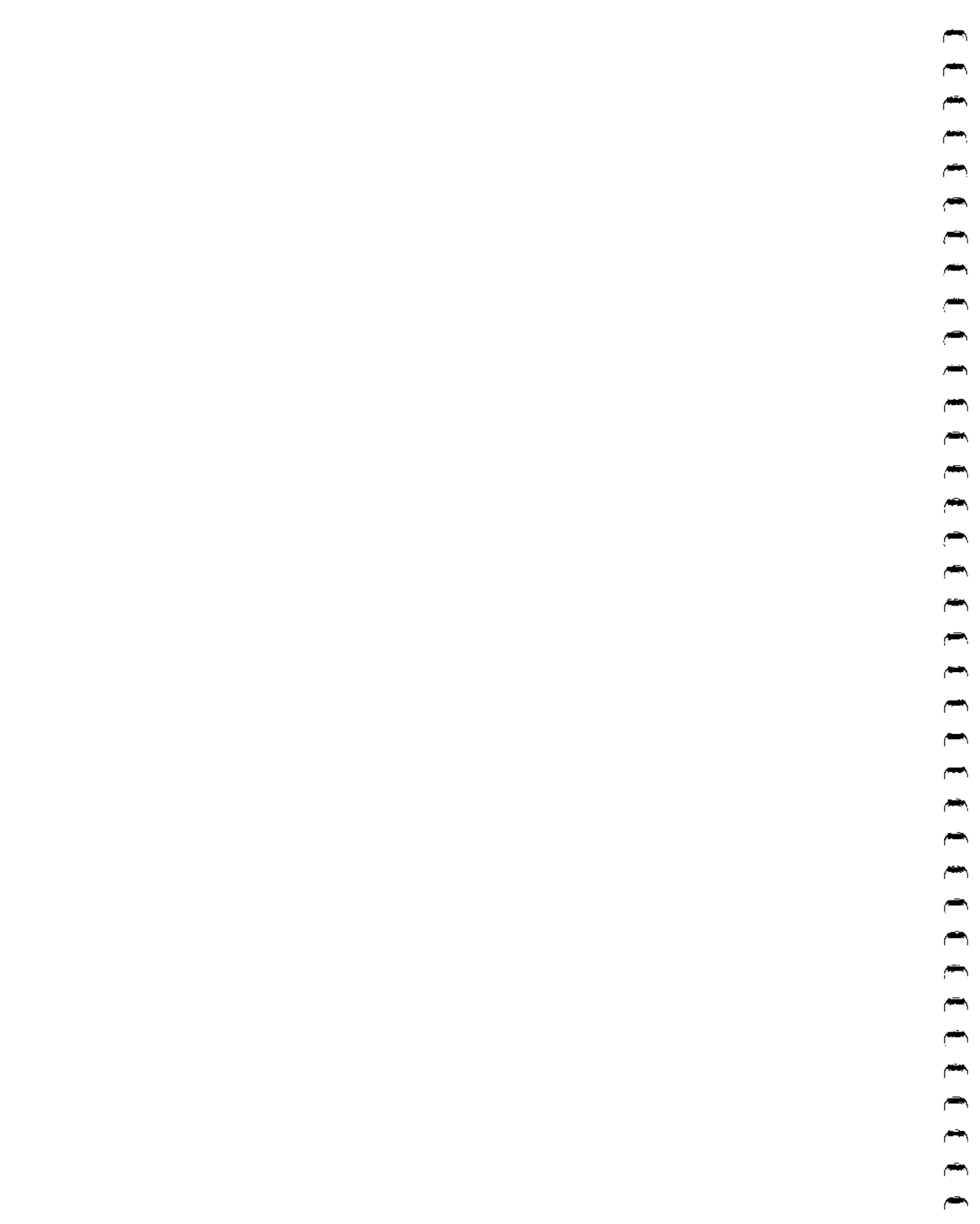
- 0** The utility executed successfully and all requested changes were made.
- >0** An error occurred.

Consequences of errors:

If the *mode* operand isn't valid, **chmod** doesn't change the file mode bits of any file.

See also:

chgrp, chown, ls, umask



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

Options:

- `-R` Recursively change ownership of files. For each *file* operand that names a directory, **chown** changes the user ID of the directory and all files in the file hierarchy below it.
- owner* A username from the user database, or a numeric user ID. The **chown** utility changes the user ID number associated with each *file* to the user ID of the *owner* specified.
- file* The pathname of a file whose ownership is to be modified.

Examples:

```
chown 27 data
chown dtdodge data
```

Description: The **chown** utility lets you change the user ownership of one or more files. For each file you name, **chown** sets the file's user ID to the user ID specified by the *owner* operand.

Exit status:

- 0 The utility executed successfully and all requested changes were made.
- >0 An error occurred.

Consequences of errors: If you invoke **chown** with the `-R` option, and **chown** attempts but fails to change the user ID of a particular file in a specified file hierarchy, it continues to process the remaining files in the hierarchy. The **chown** utility can fail to change the user ID of a file if you don't have appropriate permissions.



Using **chown** on a file clears the file's *set-group-id* and *set-user-id* bits.

See also: `chgrp`, `chmod`



Syntax: `cksum [file...]`

Options: *file* The pathname of a file to be checked. If no files are specified, the standard input is used.

Description: The **cksum** utility writes one line to standard output for each file you specify. The line contains a 32-bit CRC (cyclical redundancy check), the number of bytes in the file, and the filename, in that order.

If you don't specify any files, **cksum** processes standard input; no filename is given in the output line.

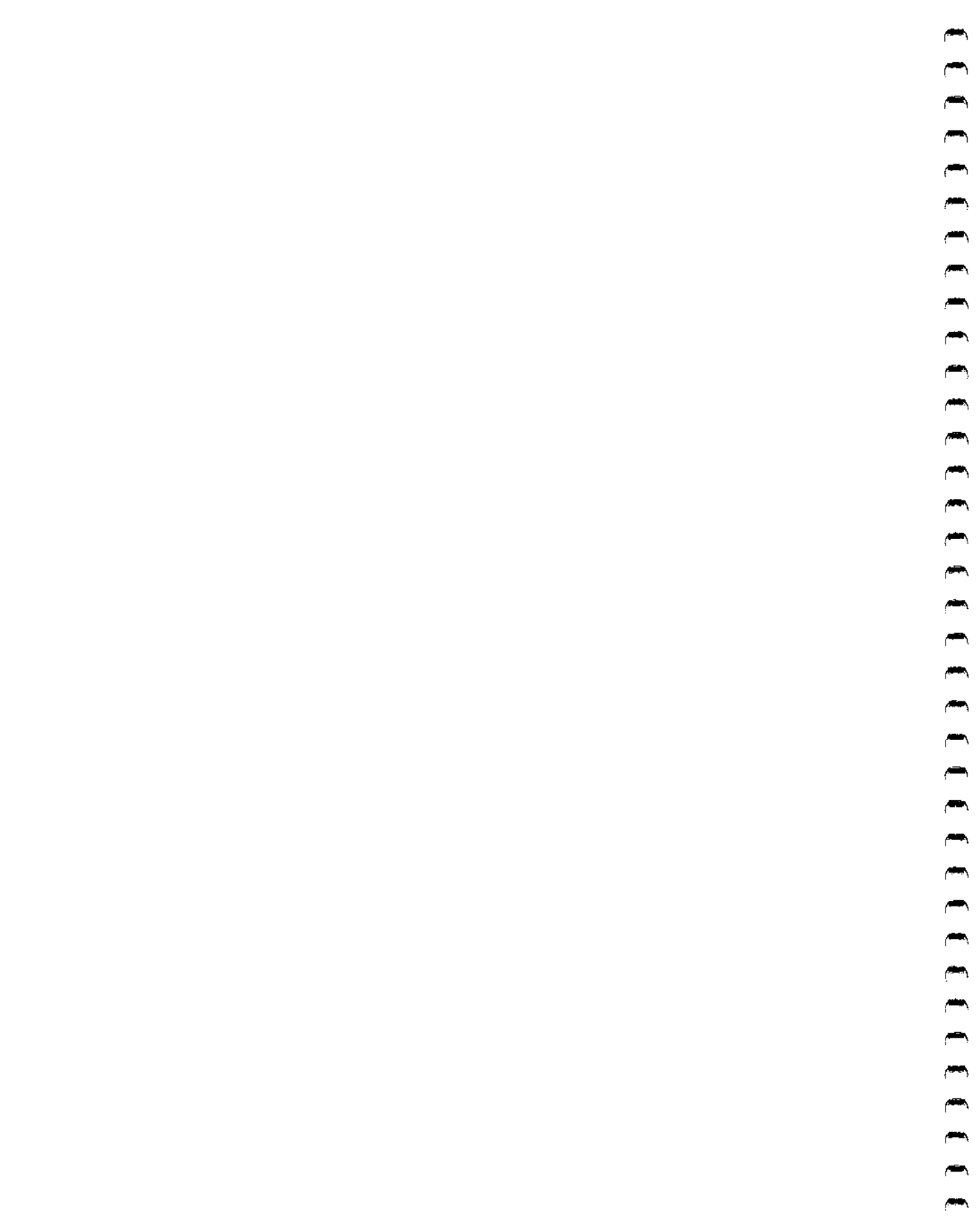
The **cksum** utility lets you quickly compare a suspect version of a file to a trusted version of the same file. You can also use **cksum** to check files after they have been transferred by modem, restored from backup media, or unpacked from a compressed form. The utilities that perform these operations have their own checks, but **cksum** serves as a useful independent checking mechanism.

If you wish to perform a byte-by-byte comparison of files, you can use the **cmp** utility.

Exit status: 0 All files were processed successfully.
>0 An error occurred.

Files: The input files can be any filetype. There are no output files.

See also: **cmp**



Syntax: **clock** [-u] [-f *foreground_color*]
 [-b *background_color*] [+format]

Options: **-u** Display Coordinated Universal Time (UTC). UTC is the standard term for Greenwich Mean Time (GMT).

-f *foreground_color* Set foreground color, where *foreground_color* can be **black**, **blue**, **green**, **red**, **cyan**, **magenta**, **brown**, **white**, **yellow**, **dgray** (dark gray), **bblue** (bright blue), **bgreen**, **bred**, **bcyan**, **bmagenta**, or **bwhite**.

 The default foreground color is white.

-b *background_color* Set background color, where *background_color* can be **black**, **blue**, **green**, **red**, **cyan**, **magenta**, **brown**, or **white**.

 The default background color is black.

+format For an explanation of how you format the date and time using the **+format** option, see the **date** utility.

Examples: Display the clock in bright cyan on blue:

```
clock -f bcyan -b blue &
```

Display the clock in green on black, in 24-hour format:

```
clock -f green "+%X" &
```

Description: The **clock** utility maintains a realtime display of the date and time in the top right corner of your console display. This utility automatically adjusts its priority to level 1 so that it doesn't compete with other tasks.

clock

display date and time on console (QNX)

With the exception of the **-f** *foreground_color* and **-b** *background_color* options, **clock** works much like the **date** utility—both let you format the date and time with the same **+format** option (for information on using this option, see **date**).

To set the date and time at the system level, you use the **date** utility. To set the date and time at the hardware level, you use the **rtc** utility (provided you have a battery backed-up clock/calendar).

See also: **date, rtc**

Syntax: `cmp [-l|-s] file1 file2`

Options:

- `-l` (the lowercase letter “l”) Print the byte position (in decimal) and the differing bytes (octal) for each difference between the two files.
- `-s` Be silent. Return exit status only.
- file1* The pathname of the first file to be compared. If *file1* is the dash character (-), the standard input is used.
- file2* The pathname of the second file to be compared.

Description: The **cmp** utility compares two files. If you don’t specify any options, **cmp** behaves as follows:

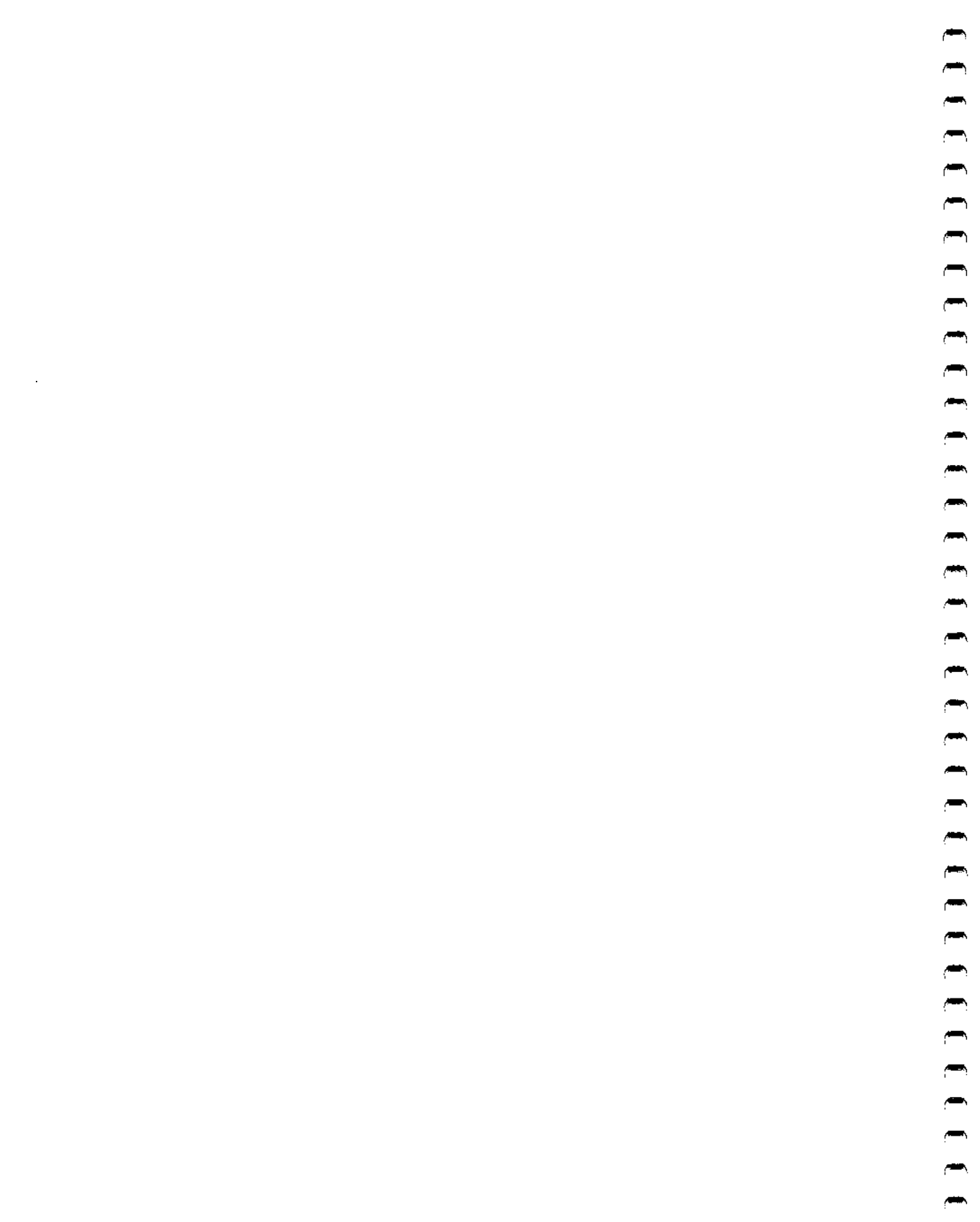
- If the two files are the same, **cmp** writes no output.
- If the files differ, **cmp** writes to standard output the byte and line number at which the first difference occurred. Bytes and lines are numbered, beginning at 1.

If you specify both the `-s` and `-l` options, nothing is printed (no long output).

Exit status:

- 0 The files are identical.
- 1 The files differ. This includes cases where one file is identical to the first part of the other. In such cases, if you haven’t specified the `-s` option, **cmp** writes to standard error a message that EOF was reached in the shorter file (before any differences were found).
- >1 An error occurred.

See also: `cksum`, `diff`



Syntax: `cp [-f|-i] [-Rp] [extensions] source_file target_file`

`cp [-f|-i] [-Rp] [extensions] source_file... target_dir`

- Options:**
- R** If the *source_file* is a directory, recursively copy the directory with the files and subdirectories under it.
 - f** Use the **-f** option to “force” **cp** to unlink the destination file prior to copying. This option prevents interactive prompting (unless **-i** is also specified) but does not disable diagnostic messages.
 - i** The **-i** option is used when you want to avoid accidentally clobbering files when copying. You will always be prompted for confirmation when the destination path exists, regardless of whether you have write permission for the destination file. When you don’t have write permission for the destination file and you respond affirmative to the prompt, the destination file will be unlinked first. Otherwise, the destination is simply overwritten and truncated.

The combination of **-i** and **-f** works as if only **-i** were specified, except that when you respond affirmative to the prompting, the destination is always unlinked first—even if you have write permission for the destination file. When only **-i** is specified, the destination is unlinked only when you do not have write permission for the destination file.

- p** After copying, attempt to duplicate the modification time of each input file in the corresponding output file. If the process does not have the appropriate privileges, the duplication will fail.

source_file The pathname of a file to be copied. If you want *source_file* to name a directory, you must also specify the **-R** option.

target_file The pathname to which a single file is copied.

target_dir The pathname of an existing directory that is to contain the output file(s).

- QNX extensions:**
- B** Use a small copy buffer (2K) instead of grabbing a large buffer (16K). If you don't specify this option and there isn't enough free memory to use a 16K buffer, **cp** dynamically reduces its buffer by 1K increments until there's enough memory available (down to a minimum of 1K).

A larger buffer is faster, especially across the LAN, but in cases where you're almost out of memory, specifying the **-B** option can mean the difference between being able to copy and having **cp** fail to open files across the network because no memory is available for the virtual circuits.

- D** This option, when used in conjunction with **-R**, controls whether **cp** will or will not descend past device boundaries while traversing a directory tree. When the **POSIX_STRICT** environment variable is set, **cp** will ignore device boundaries (i.e. will copy everything). When **POSIX_STRICT** isn't set, **cp** will by default *not* descend past device boundaries. Specifying **-D** will cause **cp** to descend past device boundaries. For example:

```
cp -R //1/ //2/backup
```

will, when **//1/** is a local hard disk, cause the contents of the disk to be backed up, but the **//1/dev** directory will be skipped, since it does not exist on the hard disk device. The addition of **-D**, as follows:

```
cp -RD //1/ //2/backup
```

would cause **cp** to also attempt to back up the contents of the **//1/dev** directory. Note that in this particular example only the disk devices (block special files) would actually have their data backed up to files in **//2/backup/dev** because **cp** won't copy character special files on recursive copies.

- l** Attempt to preserve links. When **cp** encounters a file that has a link count >1, it remembers that file's device ID. If during the **cp** another file with a link count >1 is found matching the serial number and device ID, **cp** creates a link instead of making a second copy of the file. When the destination media is changed, **cp** wipes its memory of links encountered to that point. (This is

significant when making floppy backups, or backups to removable hard disks.)

-P *[!]pat[,(!)pat]...*

Copy only files matching these patterns (where *pat* is a filename matching pattern as per the *fnmatch()* function; for details, see the section on “Pattern matching special characters”). If you specify a pattern with a preceding **!**, and if a file you’ve specified matches that pattern, **cp** doesn’t copy the file.

-v Be extra verbose. This option does everything **-v** does; it also displays lines when **cp** skips a file or a directory (i.e. you can see what **cp** isn’t doing as well as what it is doing). For example, if you select options **-R** and **-n**, you’ll find that **cp -vRn** is more useful than **cp -vRn** since option **-v** in this case might let **cp** go away and put you back at the prompt without providing you with any feedback.

-x Copy only if the destination file doesn’t exist.

-a *date/time*

Copy only those files that were last modified **after** the specified *date/time*, where *date/time* is:

" [UTC] dd/mm/yy HH:MM:SS "

specified in 24 hour time (the time portion is optional). To use Coordinated Universal Time, specify **UTC**; to use the local time, don’t include **UTC**.

If you’re running **cp** from the shell, and you want to specify both the date and the time, you must use the quotation marks. If you want to specify only the date (*dd/mm/yy*), quotation marks aren’t needed.

-b *date/time*

Copy only those files that were last modified **before** the specified *date/time* (see the **-a** *date/time* option).

- c Create any directories necessary to open the destination path. For example, if the directory:

```
/usr/eric
```

doesn't exist, and you enter:

```
cp -c file /usr/eric/source/file
```

cp performs the equivalent of:

```
mkdir -p /usr/eric/source
cp file /usr/eric/source/file
```

- d Distribute source files to multiple destinations. Ordinarily, you can specify only **one** destination file or directory to **cp**. But when you specify **-d**, and **only** when you specify **-d**, **cp** treats the word “to” as a special marker separating source files and source directories from destinations. For example:

```
cp -d myprog to //61/hd/bin //73/hd/bin //15/hd/bin
```

performs the equivalent of:

```
cp myprog //61/hd/bin
cp myprog //73/hd/bin
cp myprog //15/hd/bin
```

When you don't specify **-d**, the word “to” is treated like any other filename.

- g *grp* [*grp*]

Copy only files belonging to these groups (where *grp* is a group number or a group name).

- l *n*

(the lowercase letter “e”) If *source_file* is a directory, and if option **-R** is specified, copy only *n* levels of the directory. If you specify **-l 0**, **-R** is defeated; only the current level (named files) is copied.

-m **[!]** *mode* **[, [!]** *mode* ...

Copy only files with these modes (where *mode* is expressed as an octal number or symbolically similar to the format used in the **chmod** utility). If you specify a *mode* with a preceding **!** the **cp** utility will not copy any file that has that mode. The symbolic mode is of the form:

[[agou]=[rswx]] [, symbolic_mode]

The options of the symbolic form are:

- a** user, group, and other access
- g** group access
- o** other access
- u** user access
- r** read permission
- s** set user ID or group ID when executed
- w** write permission
- x** execute permission

For example, an option of:

-m ug=rx

copies only files for which neither group nor other has execute permission, and for which other has read permission.

- n** Copy only if the source is newer than the destination (has a more recent file modified time), or if the destination doesn't already exist.
- t** Don't attempt to duplicate file time and mode if the **-p** option was specified or if the **POSIX_STRICT** environment variable is set to on.
- u** *owner* **[,owner]** ...
Copy only files belonging to these owners, where *owner* is a user number or a username.
- v** Be verbose. Display a line of explanatory text every time a file is copied or a directory is created.

-x Copy only if the destination file already exists.

Examples: Copy **file1**, **file2**, and **file3** from the current working directory to the **/usr/eric** directory:

```
cp file1 file2 file3 /usr/eric
```

Do a raw block copy from the floppy drive on node 4 to the floppy drive on node 6:

```
cp //4/dev/fd0 //6/dev/fd0
```

Perform a backup of the entire contents of the **usr** directory to floppy disks (assuming that **/f0** is a mount point for **/dev/f0**):

```
cp -Rvpc /usr /f0
```

Recursively copy the **user/eric** directory to the **/usr/ejohnson** directory:

```
mkdir /usr/ejohnson
```

```
cp -Rv /usr/eric/ /usr/ejohnson .
```

Recursively copy the **usr/eric** directory to the **/backups/eric** directory:

```
cp -Rv /usr/eric /backups
```

Note that the difference between these last two examples is that in one case a trailing **/** follows the source directory, while in the other case no **/** is used. (See the discussion about **-R** in the section on “Recursive copying.”)

Description: **First syntax form**

```
cp [-f|-i] [-Rp] [extensions] source_file target_file
```

The **cp** utility copies the contents of the source file to the destination file named by *target_file*. This first syntax form is assumed when the destination file isn't an existing directory and there is only one source file.

Second syntax form

```
cp [-f|-i] [-Rp] [extensions] source_file... target_dir
```

For each *source_file*, **cp** copies the contents of the file to a destination file in the existing directory named by *target_dir*. The destination's filename under the target directory will be the same as its basename (final path component). For example:

```
cp dir/dir/myfile /existingdir
```

copies the contents of **dir/dir/myfile** to the following:

```
/existingdir/myfile
```

This second form is assumed when the destination file is an existing directory or when more than one source file is specified.

Unless the **-R** (recursive) option is specified, **cp** refuses to copy any *source_file* that is a directory.

What **cp** does when a destination file already exists depends on the options used. When neither **-f** nor **-i** is specified, you will be prompted only if you don't have write permission for the existing destination file. When this happens, you will be asked if you want to unlink the file first. If you don't, **cp** goes on to any remaining files. You will be prompted only if **stdin** is a **tty**. Otherwise, **cp** will print a diagnostic message to **stderr** and skip that file.

If you're copying to removable media, such as a floppy or removable disk, and the media becomes full, **cp** performs the following steps:

- 1) Closes and removes the incompletely copied destination file.
- 2) Prompts you to insert a new diskette. If you want **cp** to create the necessary directory tree on the new disk and resume copying—starting with the file that was being copied when space ran out—you enter **y**. If you want **cp** to stop copying and exit, you enter **n**.

Recursive copying

When doing a recursive copy of a directory, the destination must (obviously) be a directory. However, **cp** behaves differently depending on whether the destination directory already exists.

When the destination directory already exists, **cp** copies the contents of the source directory to a destination directory, which is given the basename of the named source directory under the named destination directory. For example, if there is a directory **/dir** that contains a single file **myfile**, then the following command:

```
cp -R /dir /existingdir
```

will cause **myfile** to be copied to **/existingdir/dir/myfile**.

Note that because of the behavior of the **basename** utility, the following command:

```
cp -R /dir/. /existingdir
```

will cause **myfile** to be copied to **/existingdir/myfile**, since the basename of **/dir/.** is simply “dot” (**.**). In other words, **existingdir/./myfile** and **existingdir/myfile** are identical.

When doing a recursive copy to a destination that doesn’t yet exist (this is possible only in the *first* syntax form—one source file, one destination file), **cp** won’t tack on the basename of the source directory to the destination. For example, this command:

```
cp -R /dir /doesntexist
```

will cause **cp** to create **/doesntexist** and then copy **/dir/myfile** to **/doesntexist/myfile**.



Although this behavior may seem confusing, it’s best to think of it this way: **cp** is doing an **mv**, except that the original copy stays where it is. If you were using **mv** to move a directory to an existing directory, you would expect the directory you are moving to appear under the destination. If you were using **mv** to move a directory to a destination directory that doesn’t exist, you would expect the utility to be simply “renaming” the directory. The rules for **cp** are the same as those for the **mv** utility.

Pattern matching
special characters

A pattern matching special character that isn’t quoted is a pattern that matches the special character itself. When not quoted, such special characters have special meaning in the specification of patterns. The

pattern matching special characters and the contexts in which they have their special meaning are as follows:

- ?** matches any printable or non-printable collating element except **<newline>**
- *** matches any string, including the null string
- [*bracket_expr*]** a pattern that matches a single collating element, except that:
 - the exclamation mark (!) replaces the circumflex (^) in its role as a nonmatching list in the regular expression notation
 - the backslash is used as an escape character within bracket expressions

Note that the characters **?**, *****, and **[** are *not* special when used inside a bracket expression.

The concatenation of patterns matching a single character is a valid pattern that matches the concatenation of the single characters or collating elements matched by each of the concatenated patterns. For example, the pattern **a[bc]** matches the strings **ab** and **ac**.

The concatenation of one or more patterns matching a single character with one or more asterisks (*****) is a valid pattern. In such patterns, each asterisk matches a string of zero or more characters, up to the first character that matches the character following the asterisk in the pattern. For example, the pattern **a*d** matches the strings **ad**, **abd**, and **abcd**, but *not* the string **abc**.

When an asterisk is the first or last character in a pattern, it matches zero or more characters that precede or follow the characters matched by the remainder of the pattern. For example, the pattern **a*d*** matches the strings **ad**, **abcd**, **abcdef**, **aaaad**, and **adddd**. The pattern ***a*d** matches the strings **ad**, **abcd**, **efabcd**, **aaaad**, and **adddd**.

Exit status:

- 0 All input files were copied successfully.
- >0 An error occurred.

cp

copy files (POSIX)

Consequences of errors:

If you specify the **-R** option, and **cp** attempts but fails to copy a particular file in a specified file hierarchy, it continues to process the remaining files in the hierarchy.

Environment variables:

POSIX_STRICT Affects whether file modification times are copied, and, if set to on, causes the QNX Extension options to be disabled.

The setting of the **POSIX_STRICT** environment variable affects the **-p** and **-t** options, as follows:

POSIX_STRICT	Option	Action
set	neither -p nor -t	if destination doesn't exist, duplicate mode only
set	-p	duplicate time, mode, user ID, group ID
set	-pt	duplicate user ID, group ID
set	-t	if destination doesn't exist, duplicate mode only
unset	neither -p nor -t	duplicate time and mode
unset	-p	duplicate time, mode, user ID, group ID
unset	-pt	duplicate user ID, group ID
unset	-t	if destination doesn't exist, duplicate mode only

Files: **Input files**

If you don't specify the **-R** option, and only one source file is named, that source file may be of any filetype.

If you specify the **-R** option, or there is more than one source file, the input files specified by each *source_file* operand, including those files contained within named directories, must be either regular files, block special files, or directories.

Output files

Each newly created output file is one of the following:

- a directory that contains copies of the files and subdirectories—if any—found in the input directory
- a regular file that has the same contents as the corresponding input file.

If an existing destination names some other type of file, **cp** opens it for writing and attempts to copy the contents of the corresponding input file to it.

See also: **basename, mv**

Caveats: When copying multiple files or doing a recursive copy, the **cp** utility refuses to copy FIFO and character special files.



Syntax: `cron &`

Description: The **cron** server schedules commands to be run at specified times, without user intervention. This server supports user-specific **cron** entries, and runs continuously. Only one **cron** server can be run per filesystem that contains a **usr/spool/cron** directory. The server must be run in the background.

Commands are specified by instructions found in **crontab** files, which are accessed via the **crontab** utility.

To minimize overhead, **cron** examines the contents of the files in **/usr/spool/cron/crontabs** when it first comes up, and then re-examines only those that have been changed via the **crontab** utility.

See also: **crontab**, *Administrator's Guide*



Syntax: **crontab** [-u *user*] [*file*]

crontab [-l] [-u *user*]

crontab [-r] [-u *user*]

Options:

- l** (the lowercase letter “el”) List the **crontab** entry of the user. Without the **-u** option, the invoking user’s entry is listed.
- r** Remove a user’s **crontab** entry. Without the **-u** option, the invoking user’s entry is removed.
- u *user*** If the **-u** option is specified when submitting a new **crontab**, the new **crontab** entry replaces or creates that user’s **crontab**. If used for removing or listing existing **crontabs**, this specifies which user’s **crontab** to remove or list. This option is valid only for superusers.
- file* The pathname of a file that contains specifications for **crontab** entries (see the Description section for the format for these specifications). If no file is specified, **crontab** uses the standard input.

Examples: List my own **crontab** entry:

crontab -l

Description: The **crontab** utility creates or replaces a user’s **crontab** entry. You can input the new **crontab** entry by specifying a file that contains specifications for **crontab** entries. If you don’t specify a file, the standard input is used.

Users may use **crontab** if their names appear in the **/usr/spool/cron/cron.allow** file. If that file doesn’t exist, the file **/usr/spool/cron/cron.deny** is checked to determine whether the user should be denied access to **crontab**. If neither file exists, only the superuser is allowed to modify **crontab** entries. If **cron.allow** doesn’t exist and **cron.deny** exists and is empty, then global usage is allowed. These permission files consist of one user name per line.

crontab

schedule periodic background work (POSIX)

Each command you specify is executed from your home directory using the shell (**/bin/sh**). The **cron** utility supplies a default environment for the shell, defining **HOME**, **LOGNAME**, **SHELL** (**= /bin/sh**), and **PATH** (**= :/bin:/usr/bin**). Users who want to have their **.profile** executed must explicitly do so in their **crontab** entry.

A **crontab** entry consists of lines of six fields each. The fields are separated by blanks. The first five are integer patterns that specify the following:

- minute (0-59)
- hour (0-23)
- day of the month (1-31)
- month of the year (1-12)
- day of the week (0-6 with 0=Sunday)

Each of these patterns can be an asterisk (meaning all valid values), an element, a list of elements separated by commas, or a range separated by a hyphen (-).

An element is either a number or two numbers separated by a hyphen (meaning an inclusive range). You can specify days by two fields (day of the month and day of the week). If both are specified, both are adhered to.

As an example of specifying the two types of days, the following line:

```
0 0 1,15 * 1
```

runs a command on the first and fifteenth of each month, as well as on every Monday. To specify days with only one field, the other field should be set to *****. For example:

```
0 0 * * 1
```

runs a command only on Mondays.

The sixth field of a line in a **crontab** entry is a string that is executed by the command interpreter at the specified times. A percent character in this field—unless escaped by a backslash—is translated to a **<newline>** character.

Only the first line (up to a % or end-of-line) of the command field is executed by the command interpreter. The remaining lines are made available to the command as standard input.

Sample **crontab** entries

Invoke the **calendar** program each day just after midnight:

```
0 1 * * * calendar -
```

Display the current time on the system console every 20 minutes:

```
1,21,41 * * * * (echo -n " "; date; echo) > /dev/console
```

Clean up the UUCP work directories each weekday at 5:30 am:

```
30 5 * * 1-5 /usr/lib/uucp/uuclean
```

Remove any files under **/tmp** that haven't been modified in more than seven days:

```
0 4 * * * find /tmp -mtime +7 -exec rm -f {} ;
```



Remember to redirect the standard output and standard error of your commands.

Exit status:

0 Successful completion.

>0 An error occurred.

See also:

cron

Caveats:

If you inadvertently enter the **crontab** utility with no argument, don't try to get out by typing <Ctrl>-<D> (end-of-file), since this would remove all entries in your **crontab** file. In this case, you should exit by typing your interrupt character, which is typically <Ctrl>-<C>.



Syntax: `ctags files...`

Examples: Generate tags for all the C source and header files in the current directory:

```
ctags *. [ch]
```

Description: The **ctags** utility generates a file called **tags** from a group of C source files.

Each C source file is scanned for *#define* statements and function definitions. The name of the macro or function becomes the name of a tag. For each tag, a line is added to the **tags** file which contains:

- the name of the tag
- a tab character
- the name of the file containing the tag
- a tab character
- a way to find the particular line within the file

Both the **less** and **vedit** utilities have the ability to use entries in the **tags** file to locate and display a definition.

See also: **less**, **vedit**



Syntax: Display date and time:

date [-u] [+format]

Set date and time:

date [-u] dd [mm [yy [HH [MM [SS]]]]] [am|pm]

Options:

-u Perform operations using Coordinated Universal Time (UTC), instead of the default local time. UTC is the standard term for Greenwich Mean Time (GMT).

[+format] The format in which the date and time are to be displayed (see the Description below).

Examples:

The following are input/output examples of **date** used at arbitrary times in the POSIX locale:

Input: **date** "+DATE: %m/%d/%y%nTIME: %H:%M:%S"

Output: **DATE: 11/21/92**

TIME: 13:36:16

Input: **date** "+TIME: %r"

Output: **TIME: 01:36:32 PM**

The following are examples of **date** used to set the date and time:

Set the system date to 22 February 1992:

date 22 2 92

Set the system date and time to 16 May 1992, 4:30 pm:

date 16 may 1992 4 30 pm

Description:

The **date** utility displays the current date and time. You can control the format of the displayed date and time by specifying an argument beginning with the + character.

The **date** utility also lets you set the date and time at the system level (to set the date and time at the hardware level, see the **rtc** utility).

date

display or set the date and time (POSIX)

You specify the format and content of the displayed date and time using field descriptors. In the output, each field descriptor is replaced by its corresponding value (see the table below). All other characters are copied to the output without change. If you want to display a % character, you encode it as %%. The output is always terminated with a **<newline>** character.

Field Descriptors:

- %a** Locale's abbreviated weekday name.
- %A** Locale's full weekday name.
- %b** Locale's abbreviated month name.
- %B** Locale's full month name.
- %c** Locale's appropriate date and time representation.
- %d** Day of the month as a decimal number (01-31).
- %D** Date in the format *mm/dd/yy*.
- %h** A synonym for **%b**.
- %H** Hour (24-hour clock) as a decimal number (00-23).
- %I** Hour (12-hour clock) as a decimal number (01-12).
- %j** Day of the year as a decimal number (001-366).
- %m** Month as a decimal number (01-12).
- %M** Minute as a decimal number (00-59).
- %n** Insert a **<newline>** character.
- %p** Locale's equivalent of either am or pm.
- %r** 12-Hour clock time (01-12) using the *AM/PM* notation in the format *HH:MM:SS AM* or *HH:MM:SS PM*.
- %S** Seconds as a decimal number (00-61). The range of values for **%S** extends from 0 to 61 seconds to accommodate the occasional leap second or double leap second.
- %t** Insert a **<tab>** character.
- %T** 24-Hour clock time (00-23) in the format *HH:MM:SS*.
- %U** Week number of the year (Sunday as the first day of the week) as a decimal number (00-53).
- %w** Weekday as a decimal number (0 (Sunday)-6).

%W Week number of the year (Monday as the first day of the week) as a decimal number (00-53).

%x Locale's appropriate date representation.

%X Locale's appropriate time representation.

%y Year without century as a decimal number (00-99).

%Y Year with century as a decimal number.

%Z Time zone name, or no characters if no time zone can be determined.

Exit status:

0 The date was displayed or set successfully.

>0 An error occurred.

Environment variables:

TZ Specifies the local time zone. The value of **TZ** affects the conversion between the system clock (UTC) and the local time.

See also: **clock, rtc**

Caveats: Some field descriptors are of unspecified format when not in the POSIX locale. As a result, parsing the output of **date** may be difficult in other locales.



Syntax: `dcheck [-B max_blks] [-L loops] [-V] [-l] [-m] [-p] [-q] [-r] [-v] [-w] [-b blk_cnt] [-f first_blk] drive`

Options:

- `-B max_blks` The maximum number of blocks to read; can be up to 32 (the default).
- `-L loops` Loop as in the `-l` option, but with a specified number of loops.
- `-V` Verify write after read.
- `-l` (the letter “e”) Loop until input (switching serial/random).
- `-m` In the bitmap, mark bad blocks as unavailable.
- `-p` Prompt before starting.
- `-r` Use a random head movement algorithm.
- `-q` Be quiet; don’t display progress information.
- `-v` Be verbose; display every bad block on the disk.
- `-w` Write after read (non-destructive) check.
- `-b blk_cnt` The maximum number of blocks to check.
- `-f first_blk` The first block to check.
- `drive` The name of the disk (e.g. `/dev/fd0`, `/dev/hd0t77`) or the root of the filesystem.

Examples: Check all blocks on the hard disk and mark bad blocks in the bitmap:

```
dcheck -m /
```

Check the first 640 blocks on the floppy disk:

```
dcheck -b 640 /dev/fd0
```

Check all blocks on the hard disk of node 3:

```
dcheck //3/dev/hd0t77
```

Description: The **dcheck** utility verifies that a disk has been correctly formatted by attempting to read every block on the drive. The block numbers of any blocks that cannot be read are displayed (in hex) to standard output. A summary of the total number of bad blocks is also

dcheck

check a disk for bad blocks (QNX)

displayed. You can use **dcheck** to check any formatted disk, including disks that contain files. The files won't be damaged.

If the number of blocks to verify is not specified, **dcheck** will obtain this information from the filesystem and will check *all* the blocks on the specified drive.

If a disk has been initialized for QNX, you should use the **-m** option to remove any bad blocks from the disk allocation bitmap (**/ .bitmap**). This is especially true for hard disks. When you specify the **-m** option, **dcheck** attempts to read the file **/ .bad_blks** from the disk. This file contains a list of all known bad blocks, in sorted order. If **/ .bad_blks** is found, **dcheck** will read it, and when it is finished checking the disk, **dcheck** will update the bitmap and recreate the **/ .bad_blks** file. Note that the **dcheck** utility only adds to, but never removes, bad block information in this file.

Some blocks may be marginal, so if you run **dcheck** multiple times (see the **-1** and **-L** options), you can increase the chance of these blocks being recognized and added to the **/ .bad_blks** file.



The **/ .bad_blks** file is also recognized by the **chkfsys** utility.

To help you find any marginal blocks, **dcheck** has a number of options to provide additional checking of a disk. For example, the **-r** option checks the blocks in a random order—each check consists of a random number of blocks between 1 and 32 (or less, depending on the value specified in the **-B** option). The **dcheck** utility keeps track of the checked blocks and checks each one only once. This option allows you to find blocks that are bad due to a slight lag time in head movement.

The **-1** option will continuously check the disk until you stop it. For this option, **-r** is implicitly used and is toggled for each invocation. That is, for the first loop, random checking is set on; for the second loop, it is off, etc. At the end of each complete check, you are prompted to stop the loop. If you don't stop it within 15 seconds, **dcheck** will be started again, etc. The **-L** option is identical to **-1** with an upper limit to the number of loops.

The **-w** option will rewrite each block on the device after reading it. This is a nondestructive check that tests the write portion of the hardware. Note that, although this is a more thorough test, you must pay for it with additional time that is dependant on the hardware.

The **-v** option is similar to the **-w** option, in that **dcheck** rewrites each block after reading it but in this case **dcheck** also rereads each block after the rewrite check and compares this second read with the first. Like the **-w** option, this test is also nondestructive. Note, however, that this is a more thorough test that takes longer.

If you wish to check a disk on another node, you must indicate the disk explicitly via a **//node** prefix.

Exit status: The exit status is the number of new bad blocks on the disk.

See also: **chkfys**, **dinit**, **fdformat**

Caveats: The **dcheck** utility normally opens the disk in read-only mode. However, if you specify the **-m**, **-w**, or **-v** options, the disk is opened in read/write mode. For read/write access, there must be no open files on the disk, or **dcheck** will fail with a “Device or resource busy” message. While **dcheck** is working in read/write mode, no other utilities or programs will be allowed to access the disk.

When using the **-m** option, if **dcheck** is terminated by a SIGBREAK or other signal, any pending bad blocks may not be recorded. In any event, the results are nondestructive.



Syntax: `df [-P] [-h] [file...]`

Options: `-P` Produce portable output (see the “Standard output” section).

file The pathname of a file within the hierarchy of the desired filesystem.

QNX extensions: `-h` Display header information.

Examples: To print “portable” information about the filesystem on which *slash* (/) resides:

```
df -P /
```

Assuming that `/usr/src` resides on the same filesystem as `/`, the following prints the same information as the above example:

```
df -P /usr/src
```

Description: The **df** utility prints the amount of available space (in kilobytes) for filesystems to which the user has appropriate read access. Filesystems are specified by the *file* operands. When no *file* operands are specified, **df** produces information for the filesystem the current directory resides on.

Standard output: When you specify the `-P` option, **df** produces output that consists of one line of information for each specified filesystem, with no preceding header lines. The fields on these lines are, in order:

filesystem name The name of the filesystem.

total space The total formatted capacity of the disk drive or partition in question.

total available user space

The total amount of space available to the user. This is the sum of *space used* + *space free*.

space used The total amount of space allocated to all files within the filesystem.

df

report disk space free (POSIX)

<i>space free</i>	The total amount of space available within the filesystem for the creation of new files by unprivileged users. When this figure is less than or equal to zero, it's not possible to create any new files on the filesystem without first deleting others.
<i>percentage used</i>	The percentage of the normally available space that is currently allocated to files. This is the fraction: $\textit{space used} / \textit{total available space}$ expressed as a percentage, rounded to the nearest integer. If <i>total available user space</i> is zero, then this percentage is displayed as "100%". This percentage may be greater than 100 if <i>total space</i> isn't equal to <i>total available user space</i>.
<i>filesystem root</i>	The directory below which the filesystem hierarchy appears.

Exit status: 0 Successful completion.
 >0 An error occurred.

See also: **du**

Syntax: `diff [-C n|-c|-e|-D string|-s] [-br] file1 file2`

Options:

- `-C n` Produce output in a form that provides *n* lines of context (see “Context output format”).
- `-b` Ignore trailing **<blank>** characters and treat other strings of **<blank>** characters as equivalent.
- `-c` Produce output in a form that provides three lines of context (see “Context output format”).
- `-e` Produce output in a form suitable as input for the UNIX line editor. The editor can then be used to convert *file1* into *file2* (see “Edit output format”).
- `-r` When *file1* and *file2* are both directories, apply **diff** recursively to files and directories of the same name (see “Directory output format”).
- file1* The pathname of a file to be compared.
- file2* The pathname of a file to be compared.

QNX extensions:

- `-D string` Produce output merging *file1* and *file2* using C Language preprocessor controls (see “C output format”).
- `-s` Produce only a summary of the differences (see “Summary output format”).

Examples: The following examples show how the **diff** utility operates on two simple files containing seven characters each:

myfile contains the following:

```
a
b
c
d
e
f
g
```


diff

report the differences between two files (POSIX)

yourfile contains the following:

```
w
a
b
x
y
z
e
```

Default output

Notice that to change **myfile** into **yourfile**, the following could be performed: insert a **w** before line 1, change lines 3 and 4 from **c d** to **x y z**, delete lines 6 and 7, which were **f** and **g**, respectively.

Invoking **diff myfile yourfile** gives the following Default output:

```
1a1
> w
3,4c4,6
< c
< d
----
> x
> y
> z
6,7d7
< f
< g
```

Edit output

Executing **diff -e myfile yourfile** gives the following Edit output:

```
0a
w
.
4,5c
x
y
z
.
8,9d
```

You may notice that the Edit output has different line numbering conventions from the Default output. This is because the script must “track” the changes it makes to the original since **ed** evaluates the line numbers based upon the current state of the file.

Description: The **diff** utility reports the differences between two files. For any two files, there may be several correct interpretations of the differences between them. The **diff** utility attempts to generate the smallest number of additions, changes, and deletions required to convert *file1* into *file2*. No output is produced if the files are identical.

The **diff** utility may generate one of the following output formats:

- default output format
- edit output format
- context output format
- directory output format
- C output format
- summary output format

Default output
format

The default output contains lines of this form:

range1 cmd range2

where *cmd* is one of: **a** (add), **c** (change), or **d** (delete).

The options *range1* and *range2* are the line numbers that differ between *file1* and *file2* respectively.

A range will consist of “startline” or “startline,endline” if more than one line is affected.

Each of the default output lines is followed by a sequence of zero or more affected lines from *file1*, preceded by the **>** character, then a sequence of zero or more affected lines from *file2* preceded by the **>** character. If lines are actually printed from both (i.e. a “change” command), then a line of three dashes (---) is output between the listing of lines from *file1* and *file2*.

Edit output format

The edit output is similar to the default output, except only the edit commands necessary to convert *file1* into *file2* are output with the relevant text.

diff

report the differences between two files (POSIX)

Context output format

The context output consists of an identification line for each file, containing the filename and modification date of the file. These lines are preceded by a line with a dozen * characters. The output lines are prefixed by either a blank character (an unaffected line, shown for context), a dash (-) character (a line to be deleted), a plus (+) character (a line to be added), or an exclamation mark (a line to be replaced).

Directory output format

For each file common to the directories that are not identical, the string:

```
"diff diff_options filename1 filename2"
```

is produced. The field *diff_options* refers to the options given to **diff** to check the two files. Each file that is present in only one directory is reported by the string:

```
"Only in directory_name:filename"
```

Subdirectories that are common to the two directories are reported as:

```
"Common subdirectories: pathname1 pathname2"
```

C output format

The C output format is a merged version of *file1* and *file2* with C preprocessor controls included so that compiling without defining *string* is equivalent to compiling *file1*. Defining *string* makes it equivalent to compiling *file2*.

Summary output format

The summary output format is one line that contains two decimal numbers separated by a single <tab> character. The numbers represent the number of lines inserted and deleted to convert *file1* into *file2*.

Exit status:

- 0 No differences were found.
- 1 Differences were found.
- >1 An error occurred.

Caveats: The **diff** utility doesn't always generate the minimum set of differences between two files.

The utility does, however, always generate a set of differences that may be used by a program to derive one file from another.

When you use the **-D** option, **diff** simply inserts preprocessor directives without regard for existing directives, which it may contradict.

See also: **cksum**, **cmp**



Syntax: `dinit [-L label] [-bhpqr] [-l label] drive`

Options:

- `-L label` Write volume *label* only to disk.
- `-b` Write bootstrap loader only to disk.
- `-h` Initialize the hard disk. You can't initialize a hard disk unless you specify this option.
- `-l label` Write volume *label* to disk after initializing.
- `-p` Pause for a keystroke before continuing.
- `-q` Be quiet; don't echo or question.
- `-r` Write root block only to disk.
- drive* Drive on which the hard disk or diskette is to be initialized (e.g. `/dev/fd0`, `/dev/hd0t77`, etc.).

Examples:

- `dinit -h /dev/hd0t77` Initialize a hard disk.
- `dinit /dev/fd0` Initialize a floppy disk.
- `dinit -hp /dev/hd0t77` Pause before initializing hard disk.
- `dinit //3/dev/fd0` Initialize floppy disk on node 3.

Description: The **dinit** utility initializes a formatted diskette or hard disk so that it may be used as a QNX filesystem. The default values are determined from the current configuration of the specified drive.

If the disk is a hard disk, you need to specify the **-h** (hard) option. This option helps protect you against typing errors that might cause **dinit** to initialize your hard disk.

Once you initialize a hard disk with **dinit**, you should use the **dcheck** utility to remove any bad blocks from the disk allocation bitmap. For example:

```
dinit -h /dev/hd0t77
dcheck -m /dev/hd0t77
```

To initialize a disk on another node, you must explicitly indicate the disk via a `//node` prefix.

A disk may be labeled with a user's specified label of up to 48 characters. You specify the label using the **-l** (lowercase letter "l") option as follows:

dinit

disk initialization (QNX)

```
dinit -l "Transfer floppy" /dev/fd0
```

If you want to label a disk that has already been initialized with **dinit**, but you don't want to re-initialize it, you may use the **-L** option.

Labels are most useful on removable media (hard disk cartridges or floppy disks) for verifying that you have the correct disk in the drive. If the disk is already labeled, you won't be allowed to re-initialize the disk without specifying a matching label or first removing the label. The label may be removed by using **-L** with an empty string as follows:

```
dinit -L "" /dev/fd0
```

The **fdformat** utility will also check for already labeled diskettes in the same manner, preventing accidental reformatting of your floppies.

When **dinit** initializes a disk, it writes a loader in the first block. If the disk is a floppy diskette, the loader is the bootstrap loader, else it is the secondary (or partition) loader. If you feel the need to rewrite the loader without re-initializing the disk, you may specify the **-b** option.

Exit status: 0 Successful.
 >0 An error occurred.

Files: All files are lost on the specified disk.

See also: **dcheck, fdformat, fdisk, mkdir**

Caveats: Don't use the **-r** option unless you know exactly what you're doing. You use the **-r** option only after a disaster has destroyed the first few blocks of your disk (e.g. a power failure occurred while the disk was being updated). In order for any damage to be repaired, you must follow **dinit -r** with this command:

```
chkfsys drive
```

dirname

return directory portion of pathname (POSIX)

Syntax: `dirname string`

Options: *string* A string representing a valid pathname.

Examples: Set the shell variable **NAME** to `/usr/src/cmd`:

```
NAME=$(dirname /usr/src/cmd/cat.c)
```

Description: The **dirname** utility returns a portion of the *string* operand to standard output. The *string* operand represents a valid pathname whose format is:

directory_pathname / base_filename

The **dirname** utility writes the *directory_pathname* component to standard output.

If *string* doesn't contain a slash character (/), the equivalent pathname `./string` is used.

You usually use the **dirname** utility inside command substitution marks, `$(...)`, within shell procedures.

Exit status: 0 Successful completion.
>0 An error occurred.

See also: **basename**



Syntax: `du [-a|-s] [-pq] [file...]`

- Options:**
- a** Generate a report for each file. If you don't specify this option, **du** makes a report for each directory. The report shows the total space allocated to all files under the directory, including the directory itself.
 - p** Report the space figures in bytes (the default is 512-byte blocks). Also, ensure that **du** generates error messages when it can't process existing files (unless the **-q** option is specified).
 - q** Be quiet; suppress error messages when **du** cannot provide statistics on files, or cannot read directories.
 - s** Give the total figures for each of the specified files, rather than the totals for any subdirectories.
- file* The pathname of a file whose size is to be displayed. If you don't specify any files, the current directory is used; **du** behaves as if the filename **dot** (.) were given.

Description: The **du** utility prints the size of the file space allocated to the specified files. If you name a directory, all files in that directory are reported; subdirectories are traversed recursively. If a file has multiple links, the space allocated to that file is counted only once.

The space figures are displayed in 512-byte blocks by default. If you want **du** to print the size in bytes, you specify the **-p** option. The sizes output by **du** when you specify the **-p** option are accurate with one exception. The numbers may be slightly higher than expected because they include extent blocks that are part of filesystem overhead associated with the file, but don't contain actual data.

If you specify nondirectories, they aren't listed unless you specify the **-a** option.

Files with multiple links are reported for only one entry.

Exit status:

- 0 Successful completion.
- >0 An error occurred. This doesn't include failure to read files or directories.

See also: **df**



Syntax: `dumper [-d path] &`

Options: `-d path` The name of the directory in which postmortem dump files are written. If not specified, the default is the user's home directory.

Examples:

```
dumper &  
dumper -d /usr/dumps &
```

Description: The **dumper** utility runs in the background and provides a postmortem dump service for all processes. Whenever a program terminates abnormally, a dump of the current state of the program is written to disk. The dump filename is the same as the program name with a `.dmp` extension. For example, if the program name is `a.out` the dump will be written to the `/usr/userid/a.out.dmp` file.

You can use the `-d` option to force all dumps into a single directory rather than into the invoking user's home directory.

A dump file may be examined using the C debugger **wvideo** command as described in the *WATCOM VIDEO User's Guide*. The dump file format is described in the `</usr/lib/sys/dumper.h>` C header file.

A program may terminate in one of two ways. It may exit cleanly under its own control, returning an exit status, or it may be forcibly terminated by the receipt of a signal it is not prepared to handle. In the latter case, **dumper** will write a dump file for the following set of signals:

SIGABRT	program-called abort function
SIGBUS	parity error
SIGFPE	floating-point error or divide by zero
SIGILL	illegal instruction executed
SIGSEGV	segment violation

dumper

dump postmortem state of a program (QNX)

You can force the dump of a running program by setting one of the preceding signals, assuming that the program is not masking or handling the signal itself. For example:

```
kill -sigabrt <process id>  
slay -s sigabrt <process name>
```

See also: *WATCOM VIDEO User's Guide*

Syntax: `echo [string...]`

Options: *string* A string to be written to standard output.

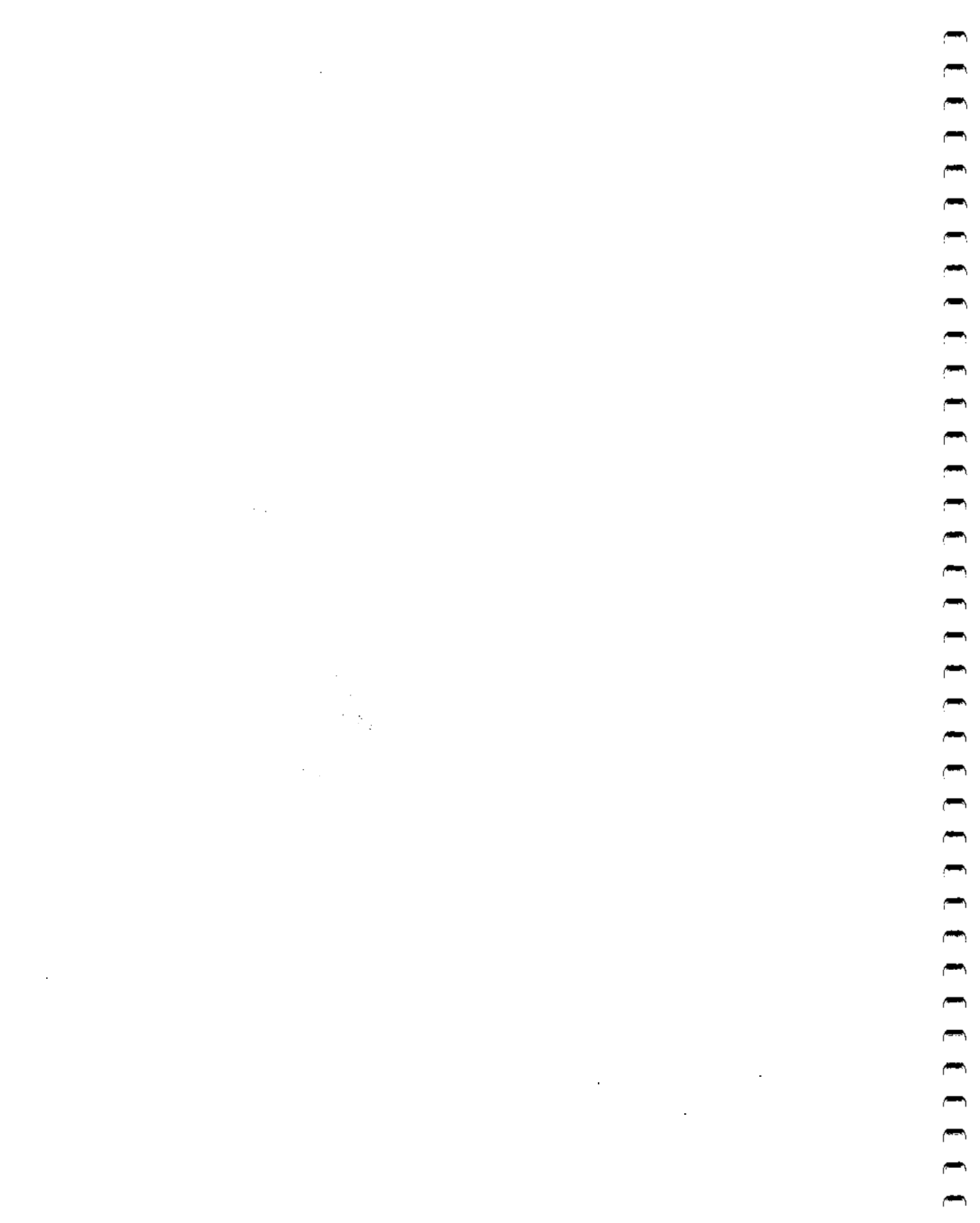
Description: The **echo** utility writes its arguments, followed by a **<newline>** character, to standard output. If there are no arguments, only the **<newline>** character is written.

The **echo** utility supports the following escape sequences within *string*:

<code>\a</code>	Write an <alert> character (the bell).
<code>\b</code>	Write a <backspace> character.
<code>\c</code>	Suppress the <newline> character that otherwise follows the final argument in the output. All characters following the <code>\c</code> in the arguments are ignored.
<code>\f</code>	Write a <form-feed> character.
<code>\n</code>	Write a <newline> character.
<code>\r</code>	Write a <carriage-return> character.
<code>\t</code>	Write a <tab> character.
<code>\v</code>	Write a <vertical tab> character.
<code>\\</code>	Write a <backslash> character.
<code>\0num</code>	Write an 8-bit value that is the 1-, 2-, or 3-digit octal number <i>num</i> .

Exit status: 0 Successful completion.
>0 An error occurred.

See also: `cat`, `sh`



Syntax: `fdformat [-I] [-L label] [-aipqv] [-h heads] [-t tracks]
 [-n sectors] [-s size] [-f first_track]
 [-l last_track] [-k skew_factor] [-z interlv]
drive`

Options:

- `-I` Ignore checking of the volume label.
- `-L label` Format only if the volume label is *label*.
- `-a` Abort on the first error encountered.
- `-f first_track` Start formatting at track *first_track*.
- `-h heads` The number of heads.
- `-i` Write layout information to the diskette, but don't format it.
- `-k skew_factor` The amount to offset the starting sector from the last sector of one track to the first sector of the next.
- `-l last_track` Stop formatting at track *last_track*.
- `-n sectors` The number of sectors per track.
- `-p` Prompt before starting.
- `-q` Be quiet; don't report progress as each track is formatted.
- `-s size` The disk media: 360K, 720K, 1.2M, or 1.4M (overrides `-h`, `-t`, and `-n`).
- `-t tracks` The number of tracks.
- `-v` Be verbose; display extra information as the format progresses.
- `-z interlv` The interleave amount.
- drive* The name of the physical device to be formatted (i.e. `/dev/fd0`).

Examples:

```
fdformat /dev/fd0
fdformat -s 1.2M /dev/fd0
```


fdformat

format floppy diskettes (QNX)

Description: The **fdformat** utility formats the specified disk. Formatting refers to a process of placing addressing marks and other control information on the disk to allow the hardware to read it. Formatting does not imply any sort of filesystem structure.

By default, **fdformat** uses the current mount characteristics of the drive as the drive parameters. These parameters may be overridden by specific command-line options, such as **-t tracks**.

The **-s size** option lets you specify the size of standard floppy disks, as in the following:

Size	Heads	Tracks	Sectors	Drive type
360K	2	40	9	5.25" floppy
720K	2	80	9	3.5" floppy
1.2M	2	80	15	5.25" floppy
1.44M	2	80	18	3.5" floppy

QNX may mark disks with a volume label. This label allows easy identification of a disk by software. By default, **fdformat** will refuse to format a disk if it encounters a volume label.

If you specify the **-L label** option, and the provided *label* matches the volume label on the disk, the format will proceed; otherwise it will fail. If you specify the **-I** option, **fdformat** will ignore the volume label information.

The first sector of QNX floppy disks contains some information about the layout of the diskette (heads, tracks, sectors). The **-i** option simply writes this layout information to a diskette without formatting it.

The **-k skew_factor** option provides important tuning for disks. Most disks have an inherent latency when moving from one track to the next. The duration of this latency depends upon the architecture of the drive.

If this latency is, for example, five sectors, you would specify the following:

-k 5

Placing the first sector of the next track five sectors away from the current sector minimizes the latency effects.

The **-z** *interlv* option lets you specify the amount of interleave (spacing between sectors). Specifying **-z1** (the default) would place the sectors contiguously (e.g. 1,2,3,4,5, etc.)

Specifying **-z2** would place the sectors at every second location (e.g. 1, -, 2, -, 3, -, 4, -, 5, etc.). If there were nine sectors per track, **-z2** would yield:

1, 6, 2, 7, 3, 8, 4, 9, 5

such that reading every other sector in the (circular) track would produce a contiguous reading of the sectors. Note that the hardware makes all of this transparent; specifying **-z** may optimize access for the hardware.

See also: **dcheck, dinit**

Caveats: The **fdformat** utility will permanently destroy any existing data on a diskette.

Note that the latency saved by tuning the disk skew factor is only realized when reading sequentially across a track boundary. On random requests, there is no single mechanism to minimize access time.



Syntax: **find** *path...* [*operand_expression*]

Description: The **find** utility recursively descends the directory hierarchy for each file specified by *path* and seeks files that match *operand_expression*.

When an operand expression isn't specified, the **find** utility will match every file and directory, printing each pathname to **stdout**, one file per line.

The operand expression follows one or more pathnames. The **find** utility treats as a pathname all the arguments up to the first one starting with any of these characters:

-
!
(

Everything after that is part of the operand expression.

Operand expressions are made up of primaries and operators. The following operators are supported:

!	(NOT)
-a	(AND)
-o	(OR)

AND operations have higher precedence than OR operators. Negation (!) has a higher precedence than AND operators. Parentheses are supported to group expressions differently.

The rules that apply to primaries and operators can be summarized as follows:

- <i>primary</i>	True if <i>primary</i> is true.
(<i>expression</i>)	True if <i>expression</i> is true.
! <i>expression</i>	(NOT) Negation of a primary or expression enclosed in parentheses.

find

find files (POSIX)

expression [-a] *expression*

(AND) True if both expressions are true. If the first expression is false, the second expression won't be evaluated. The -a is optional. AND is implied by the juxtaposition of two expressions.

expression -o *expression*

(OR) True if either expression is true. If the first expression is true, the second expression won't be evaluated.

Primary expressions

Note that when no expression is supplied, **find** behaves as if an expression of **-print** were specified.

If an expression is supplied but it contains none of the **-print**, **-ok**, **-exec**, **-spawn**, or **-ls** primaries, then the **find** utility operates as if the following expression were supplied:

(*given_expression*) **-print**

Note also that whenever a number *n* is used in a primary expression, it may be optionally preceded by a plus (+) or minus (-) sign, which changes the meaning as follows:

+*n* more than *n*

-*n* less than *n*

n exactly *n*

-name *pattern*

True if the basename of the filename being examined matches *pattern*. This follows that same pattern matching rules as used by the *fnmatch()* C function. (For details, see the section on “Pattern matching special characters” in the **cp** utility.)

-nouser True if the file belongs to a user ID that isn't in the password database.

- nogroup** True if the file belongs to a group ID that isn't in the group database.
- xdev** Always true—stops **find** from descending past directories that have a different device ID. If any **-xdev** primary is specified, it applies to the entire expression even if the **-xdev** primary wouldn't normally be evaluated.
- prune** Stops **find**'s descent from that point in the file hierarchy.
- perm [-]mode**

The *mode* argument represents file mode bits. It is identical to the *symbolic_mode* operand described in **chmod** and is interpreted as follows. To start, a template is assumed with all file mode bits cleared. An *op* symbol of:

- +** sets the appropriate mode bits
- clears the appropriate mode bits
- =** sets the appropriate mode bits, regardless of the process's file mode creation mask

An *op* symbol of **-** cannot be the first character of *mode*.

If the optional hyphen preceding *mode* is omitted, the primary is true when the file permission bits exactly match the value of the resulting template. In addition, the bits associated with the *perm* symbol **s** are ignored. If the hyphen is included, the primary is true if at least all the bits in the resulting template are set.

- perm [-]onum**

If the optional hyphen is omitted, the primary is true when the file permission bits exactly match the value of the octal number *onum* and only the bits corresponding to the octal mask 777 are compared.

find

find files (POSIX)

If the hyphen is included, more flag bits, corresponding to the octal mask 06777, are compared and the primary is true if at least all the bits in *onum* are set.

-type *c* True if the filetype is *c*, where *c* is:

- b** block special file
- c** character special file
- d** directory
- p** FIFO
- f** regular file

-links *n* True if the file has *n* links.

-user *uname*

True if the file belongs to the user *uname*. If *uname* is numeric and the name doesn't exist in the password database, *uname* is taken as a user ID. Note that *uname* is evaluated only once.

-group *gname*

True if the file belongs to the group *gname*. If *gname* is numeric and the name doesn't exist in the group database, *gname* is taken as a group ID. Note that *gname* is evaluated only once.

-size *n*[*c*]

True if the file size in bytes, divided by 512 and rounded up to the next integer, is *n*. If *n* is followed by *c*, the size is in bytes.

-atime *n* True if the file access time subtracted from the time that the **find** utility started running is between *n*-1 and *n* multiples of 24 hours.

-mtime *n* True if the file modification time subtracted from the time that the **find** utility started running is between *n*-1 and *n* multiples of 24 hours.

-ctime *n* True if the file change of status time subtracted from the time that the **find** utility started running is between *n*-1 and *n* multiples of 24 hours.

-exec *utility_name* [*argument...*] ;

True if the executed utility *utility_name* returns a zero value as exit status. The end of the primary expression is punctuated by a semicolon (;). If a *utility_name* or *argument* is { }, it is replaced by the current pathname. The current directory for the execution of *utility_name* is the same as the current directory when the **find** utility was started.

-ok *utility_name* [*argument...*] ;

Similar to **-exec**, except that **find** requests affirmation of the execution of *utility_name* using the current file as an argument by writing to standard error. If the response on standard input is affirmative, the utility is executed. If the response isn't affirmative, the command isn't executed and the value of the **-ok** operand is false.

-print Always true; causes the current pathname to be written to standard output, one pathname per line.

-newer *file* True if the current file has been modified more recently than the modification time of the file named by the *file* pathname. Note that *file* is evaluated only once.

-depth Always true; causes descent of the directory hierarchy to be done so that all entries in the directory are acted on before the directory itself. If no **-depth** primary is specified, then all entries in the directory are acted on after the directory itself. If any **-depth** primary is specified, then it applies to the entire expression even if the **-depth** primary wouldn't normally be evaluated.

find

find files (POSIX)

Primaries that are QNX extensions:

-ftime *n* True if the file creation time subtracted from the time that the **find** utility started running is between *n*-1 and *n* multiples of 24 hours.

-extents *n*
True if the file has *n* extents in the filesystem.

-ls Similar to **-print**, but displays in the same format as **ls -l** (the letter “l”).

-inode *file*
True if the file uses the same inode (has the same serial number) as the named *file*. If the file doesn’t exist and is numeric, the number is used as the serial number to match. This primary is used to find links to a file.

-spawn *cmd* [*arguments*]... ;
Similar to **-exec**, except that the command is invoked directly (i.e. not through a shell). The **-spawn** primary is faster but less flexible than **-exec**.

Examples: Search the filesystem for the **myfile** file or directory:

```
find / -name myfile
```

Remove all files named **tmp** or ending in **.xx** that haven’t been accessed for seven or more 24-hour periods:

```
find / \( -name tmp -o -name '*.xx' \) \  
-atime +7 -exec rm {} \;
```

Print the pathnames of all files in and below the current directory, but skip any directories named **SCCS** and the files beneath them:

```
find . -name SCCS -prune -o -print
```

Note that when possible, it’s better to use **find** in combination with **xargs** than it is to use the **-exec** or **-spawn** options to start commands. The **xargs** utility can be used to start a program once for many files; **-exec** or **-spawn** will start the program once for

every file matched. You'll see a tremendous difference in speed between the two approaches. For instance:

```
find / -name '*.tmp' | xargs rm
```

is generally preferable to:

```
find / -name '*.tmp' -exec rm {} \;
```

See **xargs** for more detail.

Exit status: 0 All input files were processed successfully.

 >0 An error occurred.

See also: **xargs**



Syntax: **grep** [-E|-F] [-cilnsvx]
 [-e *expression...*|-f *expression_file*|*expression*] [*file...*]

Deprecated versions:

egrep [-cilnsvx]
 [-e *expression...*|-f *expression_file*|*expression*] [*file...*]

fgrep [-cilnsvx]
 [-e *expression...*|-f *expression_file*|*expression*] [*file...*]

Options:

- E Use extended regular expression (ERE) syntax.
- F Use fixed regular expression syntax.
- c Write only a count of selected lines to standard output.
- i Ignore uppercase and lowercase distinctions during comparisons.
- l (the lowercase letter “l”) Write only the names of files containing selected lines to standard output.
- n Before each output line, display the line’s relative line number in the file.
- q Be quiet; don’t write anything to the standard output, regardless of matching lines.
- s Suppress the error messages ordinarily written for non-existent or unreadable files. Other error messages aren’t suppressed.
- v Select only those lines that don’t match the specified patterns.
- x Consider as matching lines only input lines selected against an entire fixed string or regular expression.
- e *expression*

A regular expression, of type determined by the -E, -F options. This form is used when more than one expression needs to be specified on the command line. The -e option can be repeated.

grep

search for string patterns (POSIX)

-f *expression_file*

File containing a set of regular expressions, each separated by a **<newline>**. The type of the expressions is determined by the **-E** and **-F** options. This form is used when more than one expression needs to be specified. The **-f** option can be repeated.

expression A regular expression, whose type is determined by the **-E** and **-F** options. This form is used when only one expression is specified on the command line. Any names specified after this option are treated as input files.

file The text file to be input. The default is standard input.

Examples: Display lines in **Phone.List** containing telephone numbers:

```
grep '[:digit:]\{3\}-[:digit:]\{4\}' Phone.List
```

Display all the people logged in who are listed in the **MyFriends** file:

```
who | grep -F -f MyFriends
```

Display all occurrences of the words “steve” and “barney” in the **Phone.List** file:

```
grep -F -e steve -e barney Phone.List
```

Description: The **grep** utility searches input for lines matching the expression(s) given. When an input line matches any of the expressions, it is said to be “selected.” By default, selected lines are written to standard output.

Numerous options allow variations upon the output format. For example, to reverse the meaning of the output, the **-v** option could be used.

There are three types of expressions understood by **grep**: Basic Regular Expressions, Extended Regular Expressions, and Fixed Regular Expressions. If you don’t specify **-E** or **-F**, the expression(s) are taken to be Basic Regular Expressions.

Basic and Extended Regular expressions are similar to arithmetic expressions in that larger expressions are formed by combining smaller expressions and operators according to some precedence rule.

Regular expressions have an “invisible” operator, i.e. concatenation. The concatenation of two expressions means match the one on the left, then the one on the right.

The smallest expression is a single character.

Basic regular expressions

The following table summarizes the Basic Regular Expressions, and the precedence of the operators:

<code>\ (expression \)</code>	subexpression. Match the pattern <i>expression</i> . Used for backreferences (see below), and precedence
<code>\N</code>	back-reference. Match the exact string that the <i>N</i> th subexpression did
<code>.</code>	(dot) match any single character
<code>[charset]</code>	match any member of the set <i>charset</i> (see below)
<code>c</code>	match any non-special character
<code>expression*</code>	match any number of repetitions of <i>expression</i> , including zero
<code>expression\{M\}</code>	match exactly <i>M</i> repetitions of <i>expression</i>
<code>expression\{,N\}</code>	match zero to <i>N</i> repetitions of <i>expression</i>
<code>expression\{M,N\}</code>	match <i>M</i> to <i>N</i> repetitions of <i>expression</i>
<code>expr0expr1</code>	(concatenation) match <i>expr0</i> then <i>expr1</i>
<code>^expression</code>	match <i>expression</i> only at beginning of line
<code>expression\$</code>	match <i>expression</i> only at end of line

grep

search for string patterns (POSIX)

A *charset* is formed by concatenation of the following operators:

c	any character c
c-d	any character in the range from c to d
[:alpha :]	any alphabetic character
[:upper :]	any uppercase character
[:lower :]	any lowercase character
[:digit :]	any numeric character
[:alnum :]	any numeric or alphabetic character
[:xdigit :]	any character used to represent a hexadecimal number
[:space :]	any character that is a whitespace
[:print :]	any printable character
[:punct :]	any character that is punctuation
[:graph :]	any character with a graphic representation
[:cntrl :]	any character used for control
[=a=]	an equivalence class a
[.a.]	the collating element a

If the *charset* begins with the caret (^), the set is inverted. For example:

[^[:alpha :]]

means match any non-alphabetic character.

Extended Regular Expressions

The Extended Regular Expressions are an enriched set of regular expression operators. In particular, the Extended Regular Expressions support an operator for alternation, thus allowing a match of one expression or another. It is also important to note that the parenthesis syntax is different from Basic Regular Expressions,

and the semantics are subtly different. There are no back-references in Extended Regular Expressions.

The following list summarizes the Extended Regular Expressions:

<i>(expression)</i>	match <i>expression</i> ; useful for altering precedence
.	(dot) match any single character
<i>c</i>	match any non-special character
[<i>charset</i>]	match any element of <i>charset</i>
<i>expression</i> *	match any number of repetitions of <i>expression</i> , including zero
<i>expression</i> +	match 1 to any number of repetitions of <i>expression</i>
<i>expression</i> ?	<i>expression</i> is optional (match 0 or 1 repetition)
<i>expression</i> \{ <i>M</i> \}	match exactly <i>M</i> repetitions of <i>expression</i>
<i>expression</i> \{, <i>N</i> \}	match zero to <i>N</i> repetitions of <i>expression</i>
<i>expression</i> \{ <i>M</i> , <i>N</i> \}	match <i>M</i> to <i>N</i> repetitions of <i>expression</i>
<i>expr0expr1</i>	(concatenation) match <i>expr0</i> then <i>expr1</i>
<i>expr0</i> <i>expr1</i>	(alternation) match <i>expr0</i> or <i>expr1</i> (not both)
^ <i>expression</i>	match <i>expression</i> only at the beginning of a line
<i>expression</i> \$	match <i>expression</i> only at the end of a line

Fixed Regular Expressions

Fixed Regular Expressions consist of a set of strings of characters. They do not permit the operators of Extended or Basic Regular Expressions. The algorithm used is extremely efficient for locating one of a set of strings within another string. Thus, if you don't need the various operators of Basic or Extended Regular Expressions, the Fixed Expressions are a better choice.

See also: **diff, find, sort**



Syntax: `hd [-A format] [-n count] [-s skip]
 [-t format [fmt_string]] [-v] [file...]`

-A *format* Display the file offset field in the specified format. Valid formats are:

- x** (hex, 7 digits)
- o** (octal, 10 digits)
- d** (decimal, 9 digits)
- n** (none)

-n *count* Display only *count* bytes of input.

-s *skip* Ignore the first *skip* bytes of data.

-t *format* [*fmt_string*] Use this output/display format.

-v Be verbose. Display all input.

file The pathname of an input file. If no files are specified, the standard input is used. If a *file* is a dash character (-), **hd** reads from the standard input at that point in the sequence.

Examples: Display the 2nd to 11th sectors of the floppy disk `/dev/fd0`:

```
hd -s 1b -n 10b /dev/fd0 | more
```

Description: You use the **hd** utility to output data in decimal, hex, octal, or ASCII. The name “**hd**” (hex dump) is derived from the default output format.

The **hd** utility processes input in 16-byte units that are formatted into a line. In the default output format:

- the file offset field is displayed in hex, 7 digits
- a colon and a space separates the file offset field from the data
- the data is displayed as 16 space-separated bytes in hex
- the same data is also displayed in ASCII, if printable; unprintable data appears as dots

hd

display files in decimal, hex, octal, or ASCII (QNX)

For example, if you issue the following command:

```
echo "abcdefghijklmnopqrstuvwxy01234" | hd
```

you should see the following output:

```
0000000: 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 abcdefghijklmnop
0000010: 71 72 73 74 75 76 77 78 79 7a 30 31 32 33 34 0a qrstuvwxy01234.
```

The `-v` option (verbose) inhibits `hd`'s default behavior of folding multiple identical lines into a single line that contains an asterisk (*). When `-v` is specified, all data is displayed.

To exclude part of the input, you use the `-n` and `-s` options. You can specify the arguments to these options in hex (using a `0x` prefix) or octal (using a `0` prefix).

The arguments to `-n` and `-s` also let you specify three different units:

To specify:	you add this suffix:
blocks (512 bytes)	b
kilobytes (1024 bytes)	k
megabytes (1048576 bytes)	m

To specify the output format, you use the `-t` option. The *format* argument—which can be specified in decimal, hex, or octal—instructs `hd` as to which format to use for presenting the output. The *format* argument can be one of the following:

- `o[1|2|4]` octal, 1-, 2-, or 4-byte objects (default is 2)
- `x[1|2|4]` hex 1-, 2-, or 4-byte objects (default is 2)
- `X[1|2|4]` hex 1-, 2-, or 4-bytes objects, with characters (default is 1)
- `d[1|2|4]` decimal 1-, 2-, or 4-byte objects (default is 2)
- `c` characters

The input, processed in 16-byte units formatted into a line, is displayed according to the option ([1 | 2 | 4]) you choose:

To display input as:	you choose:
sixteen 1-byte objects	1
eight 2-byte objects	2
four 4-byte values per line	4

The **x** (uppercase) format instructs **hd** to redisplay the input block at the end of line in a special character format. The special character format displays printable characters “as is” and nonprintable characters as a single dot (.). If you don’t specify the **-t** option, the default output format is the same as:

-t x

The **c** format displays printable characters as themselves. With the exception of the following characters, all other characters are displayed as 2-digit hex values.

ASCII mnemonic	Value	Representation
NUL	00	\0
<alert>	07	\a
<backspace>	08	\b
<tab>	09	\t
<newline>	0a	\n
<vertical tab>	0b	\v
<formfeed>	0c	\f
<carriage return>	0d	\r

Exit status: 0 All input files were processed successfully.
 >0 An error occurred.

See also: **cat, head, spatch**



Syntax: **head** [-c|-l] [-n *number*] [*file*...]

Options:

- c** Measure the quantity of output in bytes.
- l** (the lowercase letter “l”) Measure the quantity of output in lines; this is the default unit of measure.
- n *number*** The number of lines (default) or bytes output. The *number* argument is an unsigned decimal integer. If you omit the **-n *number*** option, the default is the first ten units (lines or bytes) of the file.
- file* The pathname of an input file. If you don’t specify any files, the standard input is used.

Examples: Display the first ten lines of all files in the current directory:

```
head *
```

Print the first 16 bytes of **myfile** in hex:

```
head -c -n 16 myfile|hd
```

Description: The **head** utility copies its input files to the standard output. The utility ends the output for each file at a point designated by the **-n *number*** option. The *number* argument is counted in bytes if you choose the **-c** option, and in lines if you choose the **-l** option.

If you specify multiple files, **head** prints an identifying header before the output for each file.

Exit status: 0 Successful completion.
 >0 An error occurred.

See also: **tail**



Syntax: `infocmp [-lCILQ] [-s {c|d|i|l|q}] [-w width]
[terminal_name]`

- Options:**
- 1** (the number one) Force the output to contain one entry per line. Without this option, output will be printed to a maximum width of 60 characters, each line containing as many entries as can fit on that line.
 - C** Use the **termcap** names for displaying the capabilities.
 - I** Use the **terminfo** names. This option is assumed by default if no others override it.
 - L** Use the long C variable names listed in `/usr/include/term.h`
 - Q** Use the names that map onto the QNX console capabilities.
 - s {c|d|i|l|q}**

Control how the output is sorted. The sort options sort the capabilities within each of the Boolean, integer, and string sections. The sort flags are:

- c** sort by the **termcap** name
- d** sort by the order stored in the **terminfo** capability file (default)
- i** sort by **terminfo** name
- l** sort by long C variable name
- q** sort by QNX capability name

-w width

Force the output to the specified number of columns. As many entries as will fit on each line will be output.

infocmp

output the contents of a terminfo capability file (UNIX)

terminal_name

The name of a terminal capability file to display. If the **TERMINFO** environment variable is *not* set, the file will be read from the **/usr/lib/terminfo** directory in a subdirectory named by the first letter of the terminal name. For example, the VT100 **terminfo** file is stored in: **/usr/lib/terminfo/v/vt100**. If set, the **TERMINFO** environment variable will define a directory where the compiled description file is read from. If a *terminal_name* is not specified, the terminal named in the **TERM** environment variable will be used.

Description: The **infocmp** utility is used to output the contents of a previously compiled **terminfo** capability file in a number of formats. The default format is suitable for editing and recompilation with the **tic** utility. The output is sorted such that the Boolean capabilities are output first, followed by the integer capabilities and then the string fields. The **infocmp** utility, executed without any options, will produce the **terminfo** description of the currently defined terminal in a form suitable for editing and recompilation with the **tic** utility.

If no options are specified and zero or one terminal name is specified, the **-I** option is assumed.

See also: **tic**, *Terminfo Technical Note*

Syntax: `kbd file`

Options: *file* The name of a keyboard layout file (typically the name of your country).

Examples: Use the German keyboard layout:

```
kbd Germany
```

Use a custom keyboard layout:

```
kbd /etc/config/kbd/custom
```

Description: The **kbd** utility is used to configure the QNX keyboard driver to respond to a particular keyboard layout. The default keyboard layout is the United States configuration. Users of QNX in other countries will use the **kbd** utility at system initialization time to specify some other keyboard layout.

Files: Keyboard layout files are typically found in the `/etc/config/kbd` directory. Custom keyboard layouts can also be used by specifying the name of a custom keyboard file.

See also: *Administrator's Guide*



Syntax: **kill** [-*signal_name*] *pid*...

kill -1

Deprecated Version:

kill [-*signal_number*] *pid*...

Options:	- <i>signal_name</i>	The name of the signal to be sent to the specified process. Values of <i>signal_name</i> are recognized in case-independent fashion, with or without the SIG prefix.
	-1	(the lowercase letter “e”) Don’t send signals. List the values of <i>signal_name</i> .
	- <i>signal_number</i>	A non-negative decimal integer representing the signal to be sent to the specified process.
	<i>pid</i>	A decimal integer specifying a process or process group to be signaled. A positive number <i>pid</i> is simply a process ID. A negative number <i>pid</i> has its absolute value taken as a process group ID. The signal is sent to all processes belonging to the group. A <i>pid</i> of zero sends the signal to all processes owned by the user in the current shell’s process group.



Don’t use a *pid* of zero when logged on as superuser. Signalling all the background processes owned by superuser and any current superuser foreground processes (e.g. backups in progress) may produce unpredictable results.

kill

terminate or signal processes (POSIX)

Examples: Any of the commands:

```
kill -9 100 -16
```

```
kill -sigkill 100 -16
```

```
kill -KILL 100 -16
```

sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose process group ID is 16, assuming the sending process has permission to send that signal to the specified processes, and that they exist.

Description: The **kill** utility sends a signal to the process(es) specified by each *pid* operand. By default, **kill** sends the SIGTERM signal, but you may override this default by naming a signal to be sent.

To print out the list of signals that may be sent, use **kill** with the **-l** option:

```
kill -l
```

Exit status: 0 At least one matching process was found for each *pid* option, and the specified signal was successfully sent to each matching process.

>0 An error occurred.

See also: **slay**

Syntax: **less** [-[+]aBcCdeEfimMnNqQrsSuUw] [-b *n*] [-x *n*]
 [-[z] *n*] [-h *n*] [-j *n*] [-p *pattern*]
 [-y *n*] [-[oO] *logfile*] [-t *tag*]
 [-T *tagsfile*] [+ *cmd*] [*file...*]

Options:



Most options may be changed while **less** is running, via the dash (-) command.

Options are also taken from the **LESS** environment variable. The environment variable is parsed before the command line, so command-line options override the **LESS** environment variable. If an option appears in the **LESS** environment variable, it can be reset to its default on the command line by using the two-character combination **++** at the beginning of the command line.

A dollar sign (\$) may be used to signal the end of an option string. This is important only for options like **-t** which take a following string.

-
- ?** Display a summary of the commands accepted by **less** (the same as the **h** command). If this option is given, all other options are ignored, and **less** exits after the help screen is viewed. (Depending on how your shell interprets the question mark, it may be necessary to quote the question mark, as follows: **-\\?**)
 - a** Start searches after the last line displayed on the screen, thus skipping all lines displayed on the screen. By default, searches begin at the second line on the screen (or after the last found line; see the **-j** option).
 - b *n*** Use a nonstandard number of buffers. Buffers are 1K, and by default 10 buffers are used (except if data is coming from standard input; see the **-B** option). The number *n* specifies the number of buffers to use.
 - B** Disable automatic allocation of buffers, so that only the default number of buffers is used. If more data is read than will fit in the buffers, the oldest data is discarded. By

less

interactive paginator (UNIX)

default, when data is coming from standard input, buffers are allocated automatically as needed to avoid loss of data.

- c Do full-screen redraws from the top line down. By default, full-screen redraws are done by scrolling from the bottom of the screen.
- C Clear the screen, then do full-screen redraws from the top line down. By default, full-screen redraws are done by scrolling from the bottom of the screen.
- d Suppress the error message normally displayed if the terminal is dumb (i.e. lacks some important capability, such as the ability to clear the screen or scroll backward). The -d option does not otherwise change the behavior of **less** on a dumb terminal.
- e Automatically exit the second time end-of-file is reached. By default, the only way to exit **less** is via the **q** command.
- E Automatically exit the first time end-of-file is reached. By default, the only way to exit **less** is via the **q** command.
- f Force nonregular files to be opened (a nonregular file is a directory or a device special file). Also suppresses the warning message when a binary file is opened. By default, **less** will refuse to open nonregular files.
- h *n* Specify a maximum number of lines to scroll backward. If it's necessary to scroll backward more than *n* lines, the screen is redrawn in a forward direction instead. (If the terminal can't scroll backward, -h 0 is implied.)
- i Ignore case; uppercase and lowercase are considered identical. Also, text that is overstruck or underlined can be searched for. This option is ignored if any uppercase letters appear in the search pattern.
- j *n* Use this line on the screen to position "target" lines. Target lines are the object of text searches, tag searches, jumps to a line number, jumps to a file percentage, and jumps to a marked position. The screen line is specified by a number: the top line on the screen is 1, the next is 2, and so on. The

number may be negative to specify a line relative to the bottom of the screen: the bottom line on the screen is `-1`, the second to the bottom is `-2`, and so on. With the `-j` option, searches begin at the line immediately after the target line. For example, if `-j4` is used, the target line is the fourth line on the screen, so searches begin at the fifth line on the screen.

- `-m` Prompt verbosely (like `more`), with the percent into the file. By default, `less` prompts with a colon.
- `-M` Prompt even more verbosely than `more`.
- `-n` Suppresses line numbers. The default (to use line numbers) may cause `less` to run more slowly in some cases, especially with a very large input file. Suppressing line numbers with `-n` will avoid this problem. Using line numbers means: the line number will be displayed in the verbose prompt and in the `=` command; the `v` command will pass the current line number to the editor.
- `-N` Display a line number at the beginning of each line.
- `-o file` Copy input to the named file as it is being viewed. This applies only when the input file is a pipe, not an ordinary file. If the file already exists, `less` will ask for confirmation before overwriting it.
- `-O file` Copy input to the named file as it is being viewed. This applies only when the input file is a pipe, not an ordinary file. If the file already exists, `less` will *NOT* ask for confirmation before overwriting it.

If no log file has been specified, the `-o` and `-O` options can be used from within `less` to specify a log file. Without a filename, they will simply report the name of the log file.

- `-p pattern`
Start at the first occurrence of *pattern* in the file. The `-p` option on the command line is equivalent to specifying `+/pattern`.

less

interactive paginator (UNIX)

-q Be moderately quiet; don't ring the terminal bell if an attempt is made to scroll past the end of the file or before the beginning of the file. If the terminal has a "visual bell," it is used instead. The bell will be rung on certain other errors, such as typing an invalid character. The default is to ring the terminal bell in all such cases.

-Q Be totally quiet; never ring the terminal bell.

-r Display "raw" control characters. The default is to display control characters using the caret (^) notation. For example, a "**control A**" (001 octal) is displayed as **^A**.

Note that when **-r** is used, **less** can't keep track of the actual appearance of the screen (since this depends on how the screen responds to each type of control character). Thus, various display problems may result, such as long lines being split in the wrong place.

-s Squeeze consecutive blank lines into a single blank line.

-S Chop rather than fold lines longer than the screen width. That is, the remainder of a long line is simply discarded. The default is to fold long lines; that is, display the remainder on the next line.

-t tag
Edit the file containing the specified tag. For this to work, there must be a file called **tags** in the current directory.

-T tagsfile
Use the specified tags file in place of the **tags** file.

-u Treat backspaces and carriage returns as printable characters (i.e. they're sent to the terminal when they appear in the input).

-U Treat backspaces and carriage returns as control characters (i.e. they're handled as specified by the **-r** option).

By default, if neither **-u** nor **-U** is given, backspaces that appear adjacent to an underscore character are treated

specially: the underlined text is displayed using the terminal's hardware underlining capability. Backspaces that appear between two identical characters are also treated specially: the overstruck text is printed using the terminal's hardware boldface capability. Other backspaces are deleted, along with the preceding character. Carriage returns immediately followed by a newline are deleted; other carriage returns are handled as specified by the `-r` option.

- `-w` Use blank lines to represent lines past the end of the file. By default, a tilde (~) character is used.
- `-x n` Set tab stops every *n* positions (the default 8).
- `-y n` Specifies a maximum number of lines to scroll forward. If it's necessary to scroll forward more than *n* lines, the screen is repainted instead. The `-c` or `-C` option may be used to repaint from the top of the screen if desired. By default, any forward movement causes scrolling.
- `-[z] n` Change the default scrolling window size to *n* lines. The default is one screenful. The `z` and `w` commands can also be used to change the window size. Note that the `z` may be omitted (as in `- n`).
- `+` If a command-line option begins with `+` (plus), the remainder of that option is taken to be an initial command to `less`. For example, `+G` tells `less` to start at the end of the file rather than the beginning; and `+/xyz` tells it to start at the first occurrence of "`xyz`" in the file.

As a special case, `+number` acts like `+numberg`; that is, it starts the display at the specified line number (note, however, that this may be slow—see the `g` command. If the option starts with two plus signs (`++`), the initial command applies to every file being viewed, not just the first one. The `+` command (described in the "Commands" section) may also be used to set or change an initial command for every file.

file A pathname of an input file. If no *file* operands are specified, `less` uses the standard input. If a *file* operand is

less

interactive paginator (UNIX)

the dash character (-), the standard input is read at that point of the sequence.

Description: The **less** utility is a program similar to **more**, but **less** allows backward movement in the file as well as forward movement. The **less** utility uses the system terminal capability database, so it can run on a variety of terminals. There is limited support for hardcopy terminals (on a hardcopy terminal, lines that should be printed at the top of the screen are prefixed with ↑).

The **less** utility will display a screen full of information, then prompt for user input by displaying a : prompt at the bottom of the screen. Commands can then be entered from the keyboard.

Commands: Commands may be preceded by a decimal number, called *n* in the following descriptions. The number is used by some commands, as indicated.

[*n*] **h**

Help: display a summary of these commands. If you forget all the other commands, remember this one.

[*n*] <spacebar> or [*n*] **f**

Scroll forward *n* lines; default is one window (see the **-z** option). If *n* is more than the screen size, only the final screenful is displayed.

[*n*] **z** Like <spacebar>, but if *n* is specified, it becomes the new window size.

[*n*] <Enter>

Scroll forward *n* lines; the default is 1. The entire *n* lines are displayed, even if *n* is more than the screen size.

[*n*] **d** Scroll forward (down) *n* lines; the default is 1/2 screen size. If *n* is specified, it becomes the new default for subsequent **d** and **u** commands.

- [*n*] **b** Scroll backward *n* lines; the default is one window (see the **-z** option). If *n* is more than the screen size, only the final screenful is displayed.
- [*n*] **w** Like **b**, but if *n* is specified, it becomes the new window size.
- [*n*] **k** Scroll backward *n* lines; the default is 1. The entire *n* lines are displayed, even if *n* is more than the screen size.
- [*n*] **u** Scroll backward (up) *n* lines; the default is ½ screen size. If *n* is specified, it becomes the new default for subsequent **d** and **u** commands.
- [*n*] **r** Redraw the screen.
- [*n*] **F** Scroll forward, and keep trying to read when the end of the file is reached. Normally this command would be used when already at the end of the file. It is a way to monitor the tail of a file which is growing while it is being viewed. (The behavior is similar to the **tail -f** command.)
- [*n*] **g** Go to line *n* in the file; the default is 1 (beginning of file). (Note that this may be slow if *n* is large.)
- [*n*] **G** Go to line *n* in the file; the default is the end of the file. Note that this may be slow if *n* is large, or if *n* isn't specified and standard input, rather than a file, is being read.
- [*n*] **p** Go to a position *n* percent into the file. *n* should be between 0 and 100. This works if standard input is being read, but only if **less** has already read to the end of the file. This is always fast, but not always useful.
- [*n*] **{** If a left brace (**{**) appears in the top line displayed on the screen, the **{** command will go to the matching right brace, which is positioned on the bottom line of the screen. If there is more than one left brace on the top line, a number *n* may be used to specify the *n*th brace on the line.
- [*n*] **}** If a right brace (**}**) appears in the bottom line displayed on the screen, the **}** command will go to the matching left brace, which is positioned on the top line of the screen. If

less

interactive paginator (UNIX)

there is more than one right brace on the top line, a number *n* may be used to specify the *n*th brace on the line.

- [*n*] (Like {, but applies to parentheses rather than braces.
- [*n*]) Like }, but applies to parentheses rather than curly braces.
- [*n*] [Like {, but applies to square brackets rather than braces.
- [*n*]] Like }, but applies to square brackets rather than braces.

<Esc> <Ctrl>-<F> *charchar*

Acts like {, but uses the two characters as open and close brackets, respectively. For example, **ESC ^F < >** could be used to go forward to the > which matches the < in the top displayed line.

<Esc> <Ctrl>- *charchar*

Acts like }, but uses the two characters as open and close brackets, respectively. For example, **ESC ^B < >** could be used to go backward to the < which matches the > in the bottom displayed line.

m *char* Followed by any lowercase letter, marks the current position with that letter.

' *char* (single quote) Followed by any lowercase letter, returns to the position that was previously marked with that letter. Followed by another single quote, returns to the position at which the last “large” movement command was executed. Followed by a ^ or \$, jumps to the beginning or end of the file respectively. Marks are preserved when a new file is examined, so the ' command can be used to switch between input files.

[n] /pattern

Search forward in the file for the *n*th line containing the pattern; *n* defaults to 1. The pattern is a regular expression, as recognized by **sed**. The search starts at the second line displayed (but see the **-a** and **-j** options, which change this).

Certain characters are special if entered at the beginning of the pattern; they modify the type of search rather than become part of the pattern:

- !** Search for lines that do *NOT* match the pattern.
- *** Search multiple files. That is, if the search reaches the end of the current file without finding a match, the search continues in the next file in the command-line list.
- @** Begin the search at the first line of the first file in the command-line list, regardless of what is currently displayed on the screen or the settings of the **-a** or **-j** options.

[n] ?pattern

Search backward in the file for the *n*th line containing the pattern. The search starts at the line immediately before the top line displayed.

Certain characters are special as in the **/** command:

- !** Search for lines that do *NOT* match the pattern.
- *** Search multiple files. That is, if the search reaches the beginning of the current file without finding a match, the search continues in the previous file in the command-line list.
- @** Begin the search at the last line of the last file in the command-line list, regardless of what is currently displayed on the screen or the settings of the **-a** or **-j** options.

less

interactive paginator (UNIX)

[*n*] <Esc> /*pattern*

Same as /*.

[*n*] <Esc> ?*pattern*

Same as ?*.

[*n*] **n** Repeat previous search, for *n*th line containing the last pattern. If the previous search was modified by **!**, the search is made for the *n*th line *NOT* containing the pattern. If the previous search was modified by *****, the search continues in the next (or previous) file if not satisfied in the current file. There is no effect if the previous search was modified by **@**.

[*n*] **N** Repeat previous search, but in the reverse direction.

<Esc> **n** Repeat previous search, but crossing file boundaries. The effect is as if the previous search were modified by *****.

<Esc> **N** Repeat previous search, but in the reverse direction and crossing file boundaries.

:e [*filename*]

Examine a new file. If the filename is missing, the “current” file (see the **:n** and **:p** commands) from the list of files in the command line is re-examined. A percent sign (%) in the filename is replaced by the name of the current file. A pound sign (#) is replaced by the name of the previously examined file. The filename is inserted into the command-line list of files so that it can be seen by subsequent **:n** and **:p** commands. If the filename consists of several files, they are all inserted into the list of files and the first one is examined.

E Same as **:e**.

[*n*] **:n** Examine the next file (from the list of files given in the command line). If a number *n* is specified, the *n*th next file is examined.

<code>[n] :p</code>	Examine the previous file in the command-line list. If a number <i>n</i> is specified, the <i>n</i> th previous file is examined.
<code>[n] :x</code>	Examine the first file in the command-line list. If a number <i>n</i> is specified, the <i>n</i> th file in the list is examined.
<code>=</code>	Print some information about the file being viewed, including its name and the line number and byte offset of the bottom line being displayed. If possible, this also prints the length of the file, the number of lines in the file, and the percent of the file above the last displayed line.
<code>-char</code>	Followed by one of the command-line option letters, this will change the setting of that option and print a message describing the new setting. If the option letter has a numeric value (such as <code>-b</code> or <code>-h</code>), or a string value (such as <code>-P</code> or <code>-t</code>), a new value may be entered after the option letter. If no new value is entered, a message describing the current setting is printed and nothing is changed.
<code>++char</code>	Followed by one of the command-line option letters, this will reset the option to its default setting and print a message describing the new setting. The <code>++X</code> command does the same thing as <code>++x</code> on the command line. This doesn't work for string-valued options.
<code>--char</code>	Followed by one of the command-line option letters, this will reset the option to the <i>opposite</i> of its default setting and print a message describing the new setting. The <code>--X</code> command does the same thing as <code>-X</code> on the command line. This doesn't work for numeric or string-valued options.
<code>+cmd</code>	Causes the specified <i>cmd</i> to be executed each time a new file is examined. For example, <code>+G</code> causes less to initially display each file starting at the end rather than the beginning.
<code>v</code>	Print the version number of less being run.

less

interactive paginator (UNIX)

	q	Exit less .
	v	Invoke an editor to edit the current file being viewed. The editor is taken from the environment variable EDITOR , or defaults to vedit .
	! <i>shell_command</i>	Invoke a shell to run the <i>shell_command</i> given. A percent sign (%) in the command is replaced by the name of the current file. A pound sign (#) is replaced by the name of the previously examined file. !! repeats the last shell command. ! with no shell command simply invokes a shell. In all cases, the shell is taken from the environment variable SHELL , or defaults to sh .
	 <i>m shell_command</i>	<i>m</i> represents any mark letter. Pipes a section of the input file to the given shell command. The section of the file to be piped is between the current position and the position marked by the letter. <i>m</i> may also be ^ or \$ to indicate beginning or end of file, respectively. If <i>m</i> is a dot (.) or <newline>, the current screen is piped. The current screen is the minimum amount piped in any case.
Environment variables:	COLUMNS	Sets the number of columns on the screen. Takes precedence over the number of columns specified by the TERM variable.
	EDITOR	The name of the editor (used for the v command).
	LESS	Flags that are passed to less automatically.
	LESSEDIT	Editor prototype string (used for the v command).
	LINES	Sets the number of lines on the screen. Takes precedence over the number of lines specified by the TERM variable.
	SHELL	The shell used to execute the ! command, as well as to expand filenames.

TERM The type of terminal on which **less** is being run.

See also: **more**

Caveats: The **=** command reports the line number of the line at the top of the screen, but the byte and percent of the line at the bottom of the screen.

If the **:e** command is used to name more than one file and if one of the named files has been viewed previously, the new files may be entered into the list in an unexpected order.



Syntax: `lex [-itv] [filename...]`

Options:

- `-i` Generate a case-insensitive scanner.
- `-t` Write the scanner to standard output rather than to the `lex.yy.c` file.
- `-v` Print a summary of the scanner statistics.

filename The source filename, or standard input if not specified.

Description: The `lex` utility generates a C program from `lex` source code. The program generated is suitable for processing character input and is designed to interface to the `yacc` utility.

The `lex` specification contains a set of regular expressions and actions where the scanner will perform the action when the regular expression is recognized in the input stream.

The following “publics” are defined in the scanner:

<code>char yytext[];</code>	When a token is recognized in the input stream, it is placed into this string, with a terminating <code>\0</code> .
<code>int yyleng;</code>	This integer contains the length of the string <code>yytext</code> .
<code>FILE* yyout</code>	The file written to for default, or ECHO actions.
<code>FILE *yyin;</code>	The file associated with the current input.

The following “externals” are required by the scanner:

<code>int input()</code>	Return the next character from the input, or return zero to signify no more input.
<code>int output (int c)</code>	Print the char <code>c</code> on <code>yyout</code> .
<code>int unput(int c)</code>	Return the character <code>c</code> to the input stream so it will be read by a subsequent call to the <code>input()</code> function.

lex

lexical analyzer generator (UNIX)

int yywrap()

Called when **lex** reaches the end of file. If there are further files to be processed, *yywrap* should initialize *yyin* and return 0, otherwise return 1.

These routines are engineered for simplifying the most common uses of **lex** scanners and may be overridden by defining them in programs themselves.

Variations of the externals are defined in the **lex** library.

Examples:

The following **lex** program will trim trailing spaces from lines, fold multiple spaces into a single tab, and convert all lowercase text to uppercase:

```
% {  
#include <stdio.h>  
#include <ctype.h>  
% }  
% %  
[a-z] putchar(toupper(yytext[0]));  
[ ]+$  
[ ]+ putchar('\t');
```

Files: `lex.yy.c`

See also: `yacc`

M.E. Lesk and E.Schmidt, "Lex—A Lexical Analyzer Generator," *Bell Laboratories Computing Science Technical Report # 39*, October 1975

S.C. Johnson, "Yacc—Yet Another Compiler-Compiler," *Bell Laboratories Computing Science Technical Report # 32*, July 1978

Syntax: `ln [-f|-i] source_file target_file`

`ln [-f|-i] source_file... target_dir`

Options:

- `-f` Force existing destination pathnames to be removed before linking; don't prompt for confirmation.
- `-i` Be interactive; write a prompt to the standard error output requesting confirmation for each link that would overwrite an existing file.

source_file The pathname of a file to be linked.

target_file The pathname of the new directory entry to be created.

target_dir The pathname of an existing directory in which the new directory entries are to be created.

Examples: Create a link to `/usr/curious/monkey` called `gorilla` in the `/usr/george` directory:

```
ln /usr/curious/monkey /usr/george/gorilla
```

Description: The `ln` utility has two syntax forms, as follows:

First syntax form

`ln [-f|-i] source_file target_file`

For each *source_file*, the `ln` utility creates a new directory entry (link) at the destination path specified by the *target_file* operand. This syntax form is assumed when the destination path **doesn't** name an existing directory.

Second syntax form

`ln [-f|-i] source_file... target_dir`

For each *source_file*, `ln` creates a new directory entry at a destination path in the existing directory named by the *target_dir* operand.

In

create links to (aliases for) files (POSIX)

The destination path for each *source_file* is the same as its basename (final path component). For example:

```
ln dir/dir/myfile /existingdir
```

will create **/existingdir/myfile** as a link to **dir/dir/myfile**.

This second syntax form is assumed when either the destination names an existing directory, or when more than one source file is specified.

If the destination path exists, and you have write permission for the existing destination file, or if **-f** was specified, the **ln** utility will unlink the destination, then create the new link.

If you don't have write permission for an existing directory path, and **-f** is not specified, and if the standard input is a tty, the **ln** utility will prompt you for confirmation prior to unlinking the existing file. If standard input isn't a tty, the **ln** utility will write a diagnostic message to standard error, and go on to the next *source_file* without unlinking the destination file.

Exit status: 0 All the specified files were linked successfully.
 >0 An error occurred.

Asynchronous events: If a destination path exists, and **ln** is interrupted before ending, the destination path may be removed, but the new link may not be created.

See also: **cp**, **mv**

Syntax: **login** [*username*]

Description: The **login** command is used when a user either first signs on to the computer, or wishes to log in as another user. If **login** is invoked without an argument, it prompts for a username and password. If invoked with an argument, the argument is assumed to be the username.

The **login** command verifies the username and password against a password database and, if verified, starts the user session. If either the username or password is unacceptable, **login** prints the message “**Login incorrect**” and terminates. If the file **/etc/nologin** exists, and the user ID is non-zero, **login** prints the contents of this file on the user’s terminal and exits.

The **login** utility sets the user and group IDs, and the current working directory, then executes a shell according to specifications found in the password database.

If execution of the shell fails, **login** prints the message “**No Shell**” and exits. The shell is started with a dash (–) prepended to its name as argument 0. This informs the shell to perform its startup routine.

The **login** utility sets the **HOME**, **SHELL**, **TERM** (if available), and **USER** environment variables. The **login** utility also updates system accounting information.

The **login** utility may be executed only from the login shell, or when there is no session active. You can’t nest **login** commands.

Files: **/etc/passwd**
 /etc/nologin
 /etc/accllog (accounting records are appended to this file)

See also: **logout, sh**



Syntax: `logout`

Description: The `logout` command terminates a QNX session. This command can be executed only from a shell invoked by a `login` command, otherwise a message is printed and the `logout` command is ignored.

If you execute a `login` command from the `login` shell, an implied `logout` is performed.

See also: `login`



Syntax: `lp [-P name] [-c] [-d name] [-n copies] [file...]`

Options:

- `-P name` Use the specified *name* as the symbolic name of a printer. If this option isn't specified, the printer named in the environment variable **PRINTER** is used. If **PRINTER** isn't set, **lp** will use the default system printer.
- `-c` Exit only after further access to any of the input files is no longer required. Note that the QNX implementation of **lp** exits this way, whether the `-c` option is specified or not.
- `-d name` Same functionality as the `-P name` option.
- `-n copies` Write this number of copies of the files, where *copies* is a positive decimal integer.
- file* The pathname of a file to be printed. If no files are specified, **lp** uses the standard input.

Examples:

- `lp myfile` Print the file **myfile**.
- `pr file1 file2 | lp` Print multiple files with headers.

Description: The **lp** utility enqueues a file into a print spooler queue. The default printer is used, unless one is specified with the `-P name` option.

The actual writing of files to the output device may occur after **lp** successfully exits.

Exit status:

- 0 All input files were processed and output successfully.
- >0 No output device was available, or an error occurred.

Environment variables:

- PRINTER** The name of the output device or destination. If the `-P name` option is specified, the utility will use that symbolic name rather than the symbolic name specified in the **PRINTER** environment variable.

See also: `lpc, lpq, lprm, pr`



Syntax: `lpc [command [argument...]]`

Description: The `lpc` utility is a tool to control the print services. There are two sides to the printing process: the *enqueueing* of jobs and the *dequeueing* of jobs. A job is enqueued by `lp` and dequeued by the print server. The job that is currently being printed is known as the “current job.”

A command may be specified on the command line; if no commands are specified, they will be read from standard input.

Brief online help messages are also available:

`help [command...]`

The help command gives a short description of each command specified. If no command is specified, a list of the available commands is displayed.

Commands: `enable {all|printer...}`

Allow enqueueing of jobs on the specified printers.

`disable {all|printer...}`

Prevent enqueueing of jobs on the specified printers.

`start {all|printer...}`

Allow enqueueing and dequeueing of jobs on the specified printers.

`stop {all|printer...}`

Prevent enqueueing and dequeueing of jobs on the specified printers.

`status {all|printer...}`

Display the status of jobs on the specified printers.

lpc

line printer control program (QNX)

abort {**all**|*printer...*}

Abort the current print job and disable the dequeuing process on the specified printers.

clean {**all**|*printer...*}

Remove all enqueued print jobs from the specified printers.

restart {**all**|*printer...*}

Examine the status of the dequeuing process, and, if necessary, restart the process on the specified printers.

topq printer {*jobnum...*|*user...*}

Change the order of jobs for the specified printer. The order is left to right and is specified either by job number or user name. Specifying by user name means all jobs owned by that user.

move fromprinter toprinter {**all**|*jobnum...*|*user...*}

Move the specified jobs from *fromprinter* to *toprinter*. Move the “current” job if no job is specified.

pause {**all**|*printer...*}

Stop the dequeuing process on the specified printers. Allow the current job to complete.

resume {**all**|*printer...*}

Start the dequeuing process on the specified printers.

<END-OF-FILE> or **exit** or **quit**

Exit from **lpc**.

See also: **lp**, **lpq**, **lprm**

Syntax: `lpq [-l] [-P name] [-q] [-s seconds] [jobnumber...]
[user...]`

Options:

<code>-P <i>name</i></code>	Use the specified <i>name</i> as the symbolic name of a printer. If this option isn't specified, the printer named in the environment variable PRINTER is used. If PRINTER isn't set, lpq will use the default system printer.
<code>-q</code>	Print a list of the available printers.
<code>-l</code>	(the letter "el") Loop. After displaying the queue, sleep for a period of time (default is 5 seconds), then redisplay the queue.
<code>-s <i>seconds</i></code>	When looping, sleep for <i>seconds</i> rather than the default. Setting this value automatically sets looping mode.
<code><i>jobnumber</i></code> <code><i>user</i></code>	Specifying either of these causes lpq to filter out only those jobs of interest.

Examples: Display all current print jobs submitted by "**steve**":

`lpq steve`

Description: The **lpq** utility reports on the status of jobs in a printer queue.

For each job, **lpq** reports the user's name, the job identifier, the ranking in the queue, and the names of files that make up the job.

If no filename is available, the name will be displayed as standard input.

When **lpq** is set for "looping," it will continue to redisplay the entries until no entries are left to display.

Exit status:

0	Jobs of interest were discovered.
1	No jobs of interest were discovered.
>1	An error occurred.

lpq

display print queue status (QNX)

Environment variables:	PRINTER	The name of the output device or destination. If the -P name option is specified, the utility will use that symbolic name rather than the symbolic name specified in the PRINTER environment variable.
Caveats:		Since lpq is based on polling, it's possible that very fast jobs will completely escape its notice.
See also:	lp, lpc, lprm	

Syntax: `lprm [-P name] [-a] [jobnumber...] [user...]`

Options:

- `-P name` Use the specified *name* as the symbolic name of a printer. If this option isn't specified, the printer named in the environment variable **PRINTER** is used. If **PRINTER** isn't set, **lprm** will use the default system printer.
- `-a` Remove all jobs owned by this user. If the user is a superuser, then the spool queue will be emptied entirely.
- jobnumber* Specifying either of these causes **lprm** to filter out only those jobs of interest.
- user*

Description: The **lprm** utility removes a job, or jobs, from a printer spool queue. If **lprm** is invoked with no arguments, it will abandon the currently active job, provided the user invoking the **lprm** utility owns the job or has appropriate privileges.

Specifying user names is useful only to the superuser. All other users can remove only their own jobs.

Environment variables:

- PRINTER** The name of the output device or destination. If the `-P name` option is specified, the utility will use that symbolic name rather than the symbolic name specified in the **PRINTER** environment variable.

See also: `lp`, `lpc`, `lpq`



Syntax: `ls [-CFRacdilqrstu1] [-DSbhnpx] [file...]`

Options:

- C** Display multiple-column output, with entries sorted down the columns according to the collating sequence.
- F** Indicate the filetype by adding an extra character after some pathnames, as follows:

- `/` (slash) for directories
- `*` (asterisk) for executable files
- `|` (vertical bar) for FIFOs (named pipes)

- R** Recursively list all subdirectories encountered.
- a** List all files, including those which start with a dot (.). By default, these entries aren't listed.
- c** For sorting (**-t**) or printing (**-l**), use time of last change to the file's status information instead of time of last modification of the file itself.
- d** Treat directories like files—give information on the directory itself, not on the files or subdirectories it contains.
- i** For each file, print the file's serial number.
- l** (the lowercase letter "el") List in long format. This option provides most of the relevant file information such as permissions, link count, owner/group of the file, as well as the size, date, and name of the file, as follows:

```
drwxrwxrwx 7 0 0 22528 Jan 17 15:38 Csrc
-rw-rw-rw- 1 0 0 22 Feb 14 13:41 barney
-rwxrwxrwx 1 0 0 22 Feb 14 13:41 exec
-rw-rw-rw- 1 0 0 22 Feb 14 13:41 fred
drwxrwxrwx 2 0 0 23040 Feb 12 10:56 libtests
drwxrwxrwx 2 0 0 2048 Sep 28 06:39 util
```

- q** Force filename characters that aren't included in the character set classification in the current locale to be displayed as a question mark (?). This is the default if output is to a terminal.
- r** Reverse the order of the sort to get either oldest files first (if files are being sorted by time) or reverse collating sequence.

- s Display the size of the file in 512-byte blocks.
- t Sort by time modified (most recently modified first) before sorting the files by the collating sequence.
- u For sorting (-t) or printing (-l), use time of last access (i.e. last use), instead of time of last modification of the file.
- 1 (the number “one”) Force output to be one entry per line.

file The pathname of a file to be listed. If the file specified isn’t found, a diagnostic message is output on standard error. Files are displayed in command-line sequence.

QNX extensions:

- D Display directories only.
- S Don’t sort the output. This option is useful for determining the order in which entries are found in a given directory.
- b Use the size of the file for sorting and printing. Sort in descending order.
- h Display a header for the -l (“el”) -x and -n options.
- n Same as -l (“el”), except display group ID/user ID numbers instead of names.
- p Display a list of relative pathnames for all non-directory files. The files are listed one to a line. This option allows you to pass full pathnames of files to programs.
- x Display extent information for a file or directory. Extent information describes how a file is segmented on disk. This option displays information similar to the following example.

file size	# of extents	block # of 1st extent	extent size (in blks)	block # of the xblk	filename
22528	8	000686	4	000C1A	Csrc
22	1	00adc3	1	000000	barney
22	1	00ACD8	1	000000	exec
22	1	00ADC2	1	000000	fred
23040	10	004A55	4	005A91	libtests
2048	1	0013EB	4	000000	util

Note that the text descriptions of the columns are not displayed on screen.

Description:	For each file you name that isn't a directory, ls displays the file's name as well as any information requested on the file.	
	For each directory you name, ls displays the names of files contained within that directory, as well as any information requested on the files. The -d option overrides this behavior and makes ls display information on the directory itself, rather than on its contents.	
	If you specify more than one file, ls displays files that aren't directories first. Directories and non-directories are sorted separately.	
	If you don't specify a file, ls displays the contents of the current directory.	
	Specifying more than one of the -C , -1 ("el"), and -1 ("one") options isn't considered an error. The last option specified determines the output format.	
	In many environments, the ls command will be "aliased" to either ls -C or ls -CF , the two most common ls display formats. Unless the POSIX_STRICT environment variable is set to "yes," ls defaults to the multi-column output (-C option).	
	The -p option is useful for passing a list of all non-directory filenames, one filename per line, to other programs. The filenames include the full pathnames.	
Exit status:	When displaying a timestamp for a file, ls will display the date and time, unless the file is older or newer than the current date by six months (a "month" is defined as 30 days). Otherwise, ls will display the date and year.	
	0	All files were listed successfully.
	>0	An error occurred.
Environment variables:		
	COLUMNS	If this variable contains a string representing a decimal integer, it indicates the user's preferred column position width for displaying multiple-column output. The ls utility calculates how many pathname text columns to

ls

list directory contents (POSIX)

display (see **-C**) based on the width provided. If **COLUMNS** isn't set or is invalid, the number of columns displayed is determined by the type of output device.

- POSIX_STRICT** Interpret options according to POSIX specifications.
- TZ** Determines the time zone for date and time displays.

See also: **find, which, ws**

Syntax: **make** [-f *makefile*]... [-DS] [-W *target*] [-deiknpqrstu]
 [*target*...] [macro=value...]

- Options:**
- D Display the text of the *makefiles*.
 - DD Display the text of the *makefiles* and the **default.mk** file.
 - S Undo the effect of the -k option. Stop processing when a nonzero exit status is returned by a command. If both -k and -S are specified, the last one specified is used. This option overrides the presence of the k flag in the MAKEFLAGS environment variable.
 - W *target* Perform the **make** as if this target has a modification time of “right now.” This is the “What-if?” option.
 - d Display the reasons why **make** chooses to rebuild a target file. All prerequisites that are newer than the target file are displayed.
 - dd Display the dependency checks in more detail. Prerequisites that are older than the target file are displayed, as well as newer prerequisites.
 - e Cause environment variables to override macro assignments within *makefiles*.
 - f *makefile* Use the description file *makefile*. If the pathname is the dash character (-), the standard input is used. If there are multiple instances of this option, they are processed in the order specified.
 - i Ignore error codes returned by commands. This is equivalent to the special **.IGNORE:** target.
 - k If a non-ignored error occurs while executing the commands to bring a target up-to-date, abandon work on the current target, but continue with targets that don’t depend on the current target. This is the opposite of -S. If both -k and -S are specified, the last one specified is used.

make

maintain, update, and regenerate groups of programs (QNX)

- n** No-execution mode. Print commands, but don't execute them. However, lines preceded by a plus sign (+) prefix or containing the string \$(**make**) are executed. Lines beginning with an @ are printed.
- p** Write to standard output the complete set of macro definitions and target descriptions.
- q** Question mode. The **make** utility returns a zero or nonzero status code, depending on whether or not the target file is up-to-date. Targets aren't updated if this option is specified.
- r** Don't read in the default file **default.mk**. That is, clear the suffix list and don't use the default inference rules.
- s** Be silent. Don't print command lines before executing them. This is equivalent to the special **.SILENT:** target.
- t** Touch the target files, changing only the file date, rather than performing the rules to reconstruct them.
- u** Unconditionally **make** the named target(s) or default target and all prerequisites.
- target* The target to bring up-to-date. If no target is specified, **make** uses the first target defined in the first *makefile* encountered.

macro=value

Macro definition. This definition remains fixed for the **make** invocation. It overrides any regular definitions for the specified macro within the *makefiles* and from the environment. This definition is inherited by subordinate **make** sessions but acts as an environment variable for these. That is, depending on the **-e** setting, it may be overridden by a *makefile* definition.

Description:

The **make** utility is used to maintain current versions of files by resolving time dependencies. If the file that **make** is examining (i.e. the target) is older than the file(s) on which it depends (i.e. the

prerequisites), then **make** executes a list of shell commands associated with the target to create or update the target.

To control what **make** does, the utility references specifications (rules) consisting of a target, prerequisites, and a list of shell commands to be executed when a prerequisite is newer than the target. To simplify maintenance of programs, **make** has a collection of default macros and inference rules allowing it to identify the proper procedures required to update targets based on minimal information.

Here's a simple example. Suppose you have a file named **myprog.c** and you wish to compile it to produce the executable program **myprog**. To do this, you would simply enter the command:

```
make myprog
```



This works even in the absence of a makefile. The alternative of using the **cc** utility would produce an executable named **a.out** by default. Using **make** for compiling single-source executables is a convenient shortcut.

The **make** utility attempts to perform the actions required to ensure that the specified target(s) are up-to-date. A target is considered out-of-date if it is older than any of its prerequisites. The **make** utility treats all prerequisites as targets themselves and recursively ensures that they are up-to-date. The utility examines the modified times of files to determine whether they are out-of-date.

After all the prerequisites of a target are ensured to be up-to-date, and if the target is out-of-date, the shell commands associated with the target entry are executed. If no shell commands are listed for the target, it is treated as up-to-date.

make

maintain, update, and regenerate groups of programs (QNX)

The section that follows the examples is an alphabetical reference for **make** containing these topics:

- Asynchronous Events
- Default Rules and Macros (5)
- Inference Rules
- Internal Macros
- Macro Definition
- Macro Expansion
- Makefile Execution (4)
- Makefile Syntax (2)
- Parallel Make
- Recursive Make
- Special Macros
- Special Targets
- Suffix Rules
- Targets (3)
- Using Makefiles (1)



If you're a first-time **make** user, you should read the topics flagged with a number in the order given. By reading these sections, as well as the examples given below, you should be able to construct simple, but useful makefiles.

The other topics will help you create more powerful, flexible, and easier-to-maintain makefiles. If you're familiar with other **make** implementations, you should read all the topics in detail in order to use **make** to its best advantage and to avoid surprises.

Examples: **make test.o**

Typing this in the absence of a makefile (or with a makefile that doesn't mention the **test.o** file) will use the built-in suffixes list and inference rules to determine the name of the source file and, if necessary, to run the proper command to create or rebuild the **test.o** file.

Suppose you have the source files **test1.c**, **test2.c**, and **test3.c**. The first two files include the headers **<hdr1.h>** and **<hdr2.h>**, and these files are all linked together to produce the program **myprog**. Here's what a simple makefile describing this could look like:

```
# Sample makefile 1

myprog: test1.o test2.o test3.o
    cc -o myprog test1.o test2.o test3.o

test1.o: test1.c hdr1.h hdr2.h
    cc -c test1.c

test2.o: test2.c hdr1.h hdr2.h
    cc -c test2.c

test3.o: test3.c
    cc -c test3.c
```

You could compile and link **myprog** by entering:

```
make myprog
```

Or since **myprog** is the first target, you could simply enter:

```
make
```

If you wanted to see what commands need to be executed without actually executing them, you could type:

```
make -n
```

Using macros, the **myprog** makefile could be simplified to:

```
# Sample makefile 2
```

make

maintain, update, and regenerate groups of programs (QNX)

```
OBJS = test1.o test2.o test3.o
HDRS = hdr1.h hdr2.h

myprog: $(OBJS)
    cc -o example $(OBJS)

test1.o: test1.c $(HDRS)
    cc -c test1.c

test2.o: test2.c $(HDRS)
    cc -c test2.c

test3.o: test3.c
    cc -c test3.c
```

This makefile is functionally identical to the previous example. It's not significantly better, just slightly more generalized.

We can further simplify the makefile by using the built-in inference rules and macros, as follows:

```
# Sample makefile 3

OBJS = test1.o test2.o test3.o
HDRS = hdr1.h hdr2.h

myprog: $(OBJS)
    $(LINK.c) -o $@ $(LDFLAGS) $(OBJS)

test1.o test2.o: $(HDRS)
```

As you can see, this makefile is short and to-the-point. Again, this is functionally equivalent to the previous examples, except for the minor addition of the macro **\$(LDFLAGS)** (which allows simple customization as you'll see later).

Using this makefile, you can customize the compilation from the **make** command line by setting the appropriate macros:

```
make CFLAGS=-Oaxt
```

which increases the optimization level of the compile.

You can also direct the linker to produce a linkmap:

```
make LDFLAGS=-M
```

Of course, any of these macro assignments could be “hard-coded” in the makefile, but it’s often convenient to override the defaults from the command line for special needs.

Asynchronous Events

If not already ignored, **make** traps SIGHUP, SIGTERM, SIGINT, and SIGQUIT and removes the current target unless it is a directory or a prerequisite of the special **.PRECIOUS** target. After cleaning up, **make** exits if the signal received was SIGQUIT. Otherwise, **make** resets the signal handler to the default and sends itself the signal that it received. The default action is taken for all other signals.

Default Rules and Macros

The default inference rules and macros for **make** are defined in the `/usr/include/make/default.mk` file, an example of which is reproduced below.



Check your copy of `/usr/include/make/default.mk` for a complete and current definition of the defaults.

If a user-supplied makefile redefines any of the default rules or macros, the result is unspecified.

The following briefly outlines the form of macro assignments and rule specifications. For more details, see “Makefile Syntax,” “Macros,” “Targets,” and “Special Targets.”

```
macro = value

target: prerequisite...
    shell command
    shell command
    .
    .
    .
```

make

maintain, update, and regenerate groups of programs (QNX)

Here are the contents of `/usr/include/make/default.mk`:

```
SUFFIXES = .o .c .y .l .h .sh
.SUFFIXES: $(SUFFIXES)

MAKE =      make
RM =        rm -f

#      C section
CC =         cc
CFLAGS =     -O
COMPILE.c = $(CC) $(CFLAGS) -c
LINK.c =     $(CC) $(CFLAGS)

.c:
    $(LINK.c) -o $@ $< $(LDFLAGS)

.c.o:
    $(COMPILE.c) $<

#      Linker section
LD =         cc
LDFLAGS =    -b

#      Lex section
LEX =        lex
LFLAGS =
LEX.l =      $(LEX) $(LFLAGS) -t

.l:
    $(RM) $*.c
    $(LEX.l) $<
    $(LINK.c) -o $@ $*.c $(LDFLAGS)
    $(RM) $*.c

.l.c:
    $(RM) $@
    $(LEX.l) $< > $@
```

```
.1.o:
    $(RM) $*.c
    $(LEX.1) $< > $*.c
    $(COMPILE.c) -o $@ $*.c
    $(RM) $*.c

#      Shell section
.sh:
    cp $< $@
    chmod +x $@

#      Yacc section
YACC =      yacc
YFLAGS =
YACC.y =    $(YACC) $(YFLAGS)

.y:
    $(YACC.y) $<
    $(LINK.c) -o $@ y.tab.c $(LDFLAGS)
    $(RM) y.tab.c

.y.c:
    $(YACC.y) $<
    mv y.tab.c $@

.y.o:
    $(YACC.y) $<
    $(COMPILE.c) -o $@ y.tab.c
    $(RM) y.tab.c
```

Inference Rules

The **make** utility uses the suffixes of a target and its prerequisites to infer how a target can be made up-to-date. A list of suffix rules defines the commands to be executed (e.g. **cc -c** would typically be executed to turn a file ending in a **.c** suffix into a file ending in a **.o** suffix). By default, **make** contains a built-in set of inference rules, supplied in the **default.mk** file. Additional rules can be specified in the explicit *makefile*, or any of the built-in rules can be redefined.

Inference rules are intimately tied to the **.SUFFIXES:** special target. Each entry in the **.SUFFIXES:** special target defines a suffix to a

make

maintain, update, and regenerate groups of programs (QNX)

filename that may be used to build another file. The inference rules then define how to actually build one file from another. These files are related, in that they must share a common basename, but have different suffixes (e.g. **test.c** and **test.o**).

For each of the suffixes specified, **make** looks to find a suffix rule. The suffix rule to update a target with a suffix of **.s1** from a prerequisite with a suffix of **.s2** is specified as a target **.s2.s1**.

For example, if the target suffix was **.o** and the source suffix was **.c**, the rule **.c.o**: specifies the command required to create a **.o** target file from a **.c** prerequisite.

When no explicit rule is found to update a target, the inference rules are checked. The suffix of the target (**.s1**) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special target. If the **.s1** suffix exists, the suffix rules are searched in the order defined by the order of suffixes in **.SUFFIXES** for a **.s2.s1** rule. A check is made for the target name **\$*.s2**. If this exists and if the file ending in **.s2** is out-of-date with respect to the file ending in **.s1**, then the commands for the **.s2.s1** rule are executed.

If the target to be built doesn't contain a suffix, and there's no rule for the target, the single-suffix rules are checked. The single-suffix rules define how a target is to be built if **make** finds a target with one of the single suffixes appended. A rule with one suffix (**.s2**) is the definition of how to build **target** from **target.s2** (e.g. the suffix rule **.c**: will be used to invoke the commands necessary to compile and link files ending in **.c** when a C source file is specified without its suffix and no rule is found, as in "**make test**").

Internal Macros

Several dynamically maintained macros are useful as abbreviations within rules. It's best not to define them explicitly.

- \$*** The basename of the current target.
- \$<** The name of the current dependency file.
- \$@** The name of the current target.
- \$?** The names of prerequisites that are younger than the target.

The `$<` and `$*` macros are normally used for inference rules. They may not be reliable when used within explicit target command lines. The internal macros may be suffixed with **D** and **F** to specify the Directory and Filename components. For example:

```
${*D}, ${@F}
```

If there is no directory in the name, dot (.) is supplied.

Macro Definition

Macro definitions are in the form:

```
string1 = string2
```

The value of *string2* is defined as all characters up to a comment character (#) or a **<newline>** character. The **<newline>** is ignored when preceded by a backslash (\).

Macro definitions may also be in the form:

```
string1 = @file
```

The value of *string1* is defined to be the contents of *file*, ignoring empty lines and, for historical reasons, lines beginning with a dollar sign (\$) or an at-sign (@). A **<newline>** character in the file will be mapped to a single space.

Subsequent appearances of `$(string1)` or `${string1}` are replaced by *string2*. See “Macro Expansion” for more details.

Macros may be added to by using the `+=` notation. Given the examples above, the following lines:

```
OBJS1 = a.o b.o c.o
OBJS1 += m.o n.o
```

would be the same as:

```
OBJS1 = a.o b.o c.o m.o n.o
```

Macro assignments come from the sources listed below. If a macro name already exists at the time it is being processed, the newer definition replaces the existing definition unless it is marked as unchangeable. The order is as follows:

make

maintain, update, and regenerate groups of programs (QNX)

- 1) macros specified on the command line are added and marked as unchangeable
- 2) macros defined in the **make** “built-in” rules (the **default.mk** file) are used
- 3) the contents of the environment are added to the macro table in the order defined in the environment
- 4) macros defined in the explicit *makefile*(s); multiple *makefiles* are processed in the order specified

If the **-e** option is present, the order changes to:

- 1) macros specified on the command line are added and marked as unchangeable
- 2) macros defined in the **make** “built-in” rules (the **default.mk** file) are used
- 3) macros defined in the explicit *makefile*(s); multiple *makefiles* are processed in the order specified
- 4) the contents of the environment are added to the macro table in the order defined in the environment

The **SHELL** macro is treated specially. It is provided by **make** and is set to the pathname of the Shell command language interpreter (**/bin/sh**). The **SHELL** environment variable is always ignored. If **SHELL** is defined in the explicit *makefile* or is specified on the command line, then it replaces the original value.

Macro Expansion

If you have defined a macro as follows:

string1 = *string2*

then subsequent appearances of **\$** (*string1*) or **\${***string1***}** are replaced by *string2*. If *string1* is a single character, the parentheses or braces are optional.

Macros can appear anywhere in the makefile and are evaluated when used. Macros used in targets or prerequisites are evaluated when the makefile is being read. All other macros are evaluated when

commands are executed. Macros used within macros are evaluated only when the new macro itself is evaluated. For example:

```
OBJSALL = $(OBJS1) $(OBJS2)
OBJS1 = a.o b.o c.o
OBJS2 = x.o y.o z.o
```

`$(OBJSALL)` will become `a.o b.o c.o x.o y.o z.o`

The optional form `$(string1[:subst1=[subst2]])` can be used to replace all non-overlapping occurrences of *subst1* with *subst2* when the macro substitution is performed. The occurrence of *subst1* must be a suffix at the end of a word in *string1*. It is as though the **sed** expression of the following form had been used:

```
s/\(.*\)subst1\([<blank><newline>]\)\)/\1subst2\2/g
```

For example, using the macros defined above, you could create a list of source files (assuming they're all C source files) with the following:

```
SRCS = $(OBJSALL:o=c)
```

`$(SRCS)` will become `a.c b.c c.c x.c y.c z.c`

Makefile Execution

Once **make** has brought all the prerequisites up-to-date, it executes the list of specified shell commands. These commands are processed one line at a time. Command processing is done by writing the command to the standard output (except where noted otherwise) and then executing the command. Commands are executed by passing the command line to the shell in the same manner as the *system()* function.

The environment for the shell command being executed contains all of the variables in the environment of **make**. The macros from the command line to **make** and the **MAKEFLAGS** variable are added to the environment, overwriting any variable with the same name.

By default, when **make** receives a nonzero status from the execution of a shell command, it terminates with an error message to standard error.

make

maintain, update, and regenerate groups of programs (QNX)

All errors found while executing the command are ignored if any of the following conditions is true:

- the command prefix contains a hyphen (-)
- the **-i** option is present
- the special target **.IGNORE**: is in the *makefile*

The command is *not* written to standard output before it is executed if any of the following conditions are true:

- the command prefix contains an at-sign (@)
- the **-s** option is present
- the special target **.SILENT**: is in the *makefile*

If the **-n** option is present, **make** only writes the command line to standard output and does *not* execute the command. The only exceptions are if the command prefix contains a plus sign (+) or if the line contains the string **\$(MAKE)**.

The **-n** option is always added to **MAKEFLAGS**. This allows a recursive **make -n target** to be used to see all of the action that would be performed on the target.

If the **-k** option is present, **make** terminates processing the commands for the current target if an error occurs executing one of the commands, but continues working on other targets that don't depend on the current target.



If the **-t** option is present, **make** writes the command line to standard output. But instead of executing the command, **make** updates the modified time of the target as though **touch target** had been executed. (See the **touch** utility.) This can be dangerous when sources are maintained by more than one person.

For more details on command lines and command prefixes, see "Targets."

Makefile Syntax

The *makefile(s)* may contain a mixture of comment lines, macro definitions, include lines, and target lines including special targets

and suffix rules. Lines may be continued across input lines by escaping the **<newline>** with a backslash (\).

Anything after a pound sign (#) is considered a comment and is stripped from the line, including spaces immediately before the #. If the # is inside a quoted string, it isn't treated as a comment. Completely blank lines are ignored.

An include line is used to include the text of another makefile. It consists of the word **include** with no leading blanks, followed by spaces and by the name of the file to be included at this line. Macros in the name of the file being included are expanded before the file is included. Files that are included may be nested. An example of an include line is:

```
include /usr/include/make/default.mk
```

For more details on the other different line types, see the following topics: "Targets," "Macros," "Special Targets," and "Suffix Rules."

Parallel Make

Perhaps the most powerful, extended feature of the QNX **make** utility is its ability to analyze the dependency graph defined by the *makefile* and the inference rules and to identify targets that can all be made *at the same time*.

At startup, **make** examines the macro **WORKNODES** to obtain a list of nodes in the QNX network that may be used for the parallel make facility. (Although this macro may be defined in a makefile, it is usually defined as an environment variable.)

Here's an example of the macro:

```
WORKNODES = 61 15 18 67
```

The nodes may be in any order, separated by either blanks or colons.

This list of nodes is examined to eliminate duplicate node numbers and nodes that are not currently accessible. The node where **make** itself is running is also added to the list (if it wasn't already on it). The utility assumes that if you're running **make** on a node, you should be prepared to execute the rules on that node. For example, using the above macro definition, if **make** were started on node 3 and

make

maintain, update, and regenerate groups of programs (QNX)

all the nodes were accessible, the resulting list of work nodes would be: 61 15 18 67 3.

If **make** identifies an out-of-date prerequisite, it will choose the “fastest” node (using the CPU speed value printed by the utility **sin info**) that doesn’t already have a rule assigned to it. The utility will then start executing the rule on that node.

If all the work nodes are busy executing rules, **make** will wait for one of the dispatched rules to complete, thereby freeing up a node, before it submits the next rule.

After **make** has done this with all the out-of-date prerequisites, it will wait (synchronize) for the dispatched rules to finish executing before attempting to make the current target.

Since **make** not only identifies the prerequisites that may be made in parallel, but also automatically synchronizes itself, the result is an extremely powerful, useful facility usually requiring no changes to the makefile. Simply by defining **WORKNODES** in your environment, you can turn an ordinary makefile into a parallel make.

Sometimes, however, a convoluted dependency graph will be defined (usually because of an incomplete list of prerequisites in the makefile), and **make** will need some help in identifying synchronization points. If **make** ever attempts to build a target while its prerequisites are still in the process of being built (in parallel), then you should add a **%sync** line to your makefile as the first rule of the target requiring the synchronization, as shown below:

```
example.lib:    $(OBJECTS)
               %sync
               ...
```

Recursive Make

The **make** utility has the ability to invoke itself. This can be extremely useful for special circumstances, such as making a library in another directory before linking a program that requires it.

When **make** is processing commands, if the command prefix contains a plus (+) or the line contains the string **\$(MAKE)**, the command will always be executed, even if the **-n** option is present. See “Targets” for details.

Here's a commonly seen makefile construct that demonstrates the use of a recursive make:

```
program: libdir/example.lib
    ...

libdir/example.lib:
    cd $(@D) ; $(MAKE) $(@F)
```

Special Macros

MAKE	This normally has the value make . Any command that invokes make temporarily overrides the -n option, just for the duration of the one line. This allows nested invocations of make to be tested with the -n option.
MAKEFLAGS	This macro has the set of options provided to make as its value. If this is set as an environment variable, the set of options is processed before any command-line options. This macro may be explicitly passed to nested makes, but it is also available to these invocations as an environment variable. The -f and -d flags aren't recorded in this macro.
SUFFIXES	This contains the default list of suffixes supplied to the special .SUFFIXES: target. It isn't sufficient to simply change this macro in order to change the .SUFFIXES: list. That target must be specified in your <i>makefile</i> . See "Inference Rules."
\$	This macro translates to a dollar sign. Thus you can use \$\$ in the <i>makefile</i> to represent a single \$.

Special Targets

.ALT_DIRS:	Prerequisites of this special target form the list of directories to search for an implied prerequisite file if it cannot be located in the target directory. After the search for an implicit prerequisite in the target directory has been exhausted, the filename component of the target (the target name, excluding any directory portion) is appended to each prerequisite of this special target in turn, and the search for the implicit prerequisite is tried again. Subsequent occurrences of .ALT_DIRS: add to the
-------------------	--

make

maintain, update, and regenerate groups of programs (QNX)

list of alternate directories. There are no commands associated with the **.ALT_DIRS**: special target.

- .DEFAULT**: The default is used to process a target when there is no other entry for it, and no implicit rule for building it. The **make** utility ignores all prerequisites for the default target.
- .DONE**: This target and its prerequisites are processed after all other targets are built.
- .FAILED**: If **.FAILED** is defined and **make** halts with an error, then this target (and its prerequisites), is performed instead of **.DONE**:.
- .IGNORE**: Nonzero error codes returned from commands are ignored. Specifying this in a *makefile* is the same as specifying **-i** on the command line.
- .INIT**: This target and its prerequisites are processed before any other targets are processed.
- .PRECIOUS**: Prerequisites of this special target aren't removed if **make** receives an asynchronous event. Subsequent occurrences of **.PRECIOUS**: add to the list of precious files. There are no commands associated with the **.PRECIOUS**: special target.
- .SILENT**: Commands aren't echoed before they're executed. Specifying this in a *makefile* is the same as specifying **-s** on the command line.
- .SUFFIXES**: The suffixes list for selecting implicit rules. Specifying this target with prerequisites adds these to the end of the suffixes list. Specifying it with no prerequisites clears the list. In order to add your own prerequisites to the head of the list, you could enter:

.SUFFIXES:

.SUFFIXES: **.abc \$(SUFFIXES)**

Suffix Rules

Targets that begin with a period and don't contain any slashes are treated as suffix rules. If there is only one period in the target, it is a single suffix rule. Targets with two periods are double suffix rules.

Suffix rules don't have prerequisites, but do have commands associated with them. See "Inference Rules." For examples of the default rules, see "Default Rules and Macros."

Suffix rules can be redefined or canceled. A target that matches an existing suffix target overwrites the old suffix rule. If no commands are given, the rule is never used.

Targets

A target entry in the *makefile* has the following format:

```
target... : [prerequisite...] [; command]
<blank>shell_command
<blank>shell_command
.
.
.
```

Any line that doesn't have leading white space (other than macro definitions) is a "target" line. Target lines consist of one or more filenames (or macros that expand into same) called targets, followed by a colon (:). The colon is followed by a list of prerequisite files. The prerequisite list may be terminated with a semicolon (;), which may be followed by a shell command.

A line in a *makefile* that starts with a <tab> or <space> character is a shell line or rule. This line is associated with the current target entry. A sequence of commands may be associated with a single target entry. Shell lines may have any combination of the following characters to the left of the command as a *command prefix*:

- @ won't echo the command line, except if **-n** is used
- **make** will ignore the exit code of the command;
without this, **make** terminates when a nonzero exit
code is returned
- + **make** executes the command even if **-n** was specified

make

maintain, update, and regenerate groups of programs (QNX)

If a target is named in more than one target line, the prerequisites and rules are added to form the target's complete prerequisite list and rule list.

The prerequisites are those from which a target is constructed. They in turn may be targets of other prerequisites. In general, for a particular target file, each of its prerequisite files is “made” to ensure that each is up-to-date with respect to its prerequisites.

The modification time of the target is compared to the modification time of each prerequisite file. If the target is older than the prerequisite, then one or more of the prerequisites have changed, and the target must be constructed. This checking is done recursively, so that all prerequisites of prerequisites of prerequisites *ad infinitum* are up-to-date.

To reconstruct a target, **make** expands macros, strips off initial white space, and passes each rule to a shell for execution.

For target lines, macros are expanded as the *makefile* is read. Command lines have macro expansion delayed until the line is to be executed.

Using *makefiles*

The **make** utility comes with a set of built-in macro definitions and rules supplied as a default *makefile* (named **default.mk**). If the **-r** option is specified, the “built-in” rules aren't used (the default file isn't read) and the suffix list is cleared. The default *makefile* (**default.mk**) is searched for in the following order:

- 1) in the current directory
- 2) in the user's home directory
- 3) in the default directory **/usr/include/make**

If you wish to add to the default rules with a local **default.mk** file, you may use the following in your default file to include the system defaults:

```
include /usr/include/make/default.mk
```

You can specify additional rules in a user-specified *makefile*. Targets, rules, and macros of the same name are overwritten by the

user-specified file. By default, the `./makefile` file is used. If `./makefile` isn't found, the file `./Makefile` is then tried.

The `-f` option directs **make** to ignore `./makefile` and `./Makefile` and use the specified argument as a *makefile* instead. If the `-` argument is specified, standard input is used. More than one additional *makefile* can be used by specifying multiple `-f` options. Using `-f` multiple times performs a concatenation of all the *makefiles* in a left-to-right order.

See “Makefile Syntax” for more details.

Exit status: 0 Successful.

 >0 An error occurred.

Error messages: The following error messages may be displayed:

Badly formed macro

A macro definition that has incorrect syntax has been encountered. Most likely, the name is missing.

Cannot open file

The *makefile* indicated in an include directive wasn't found or wasn't accessible.

Don't know how to make target

There is no *makefile* entry for target, none of **make**'s implicit rules apply, there is no `.DEFAULT:` rule.

Improper macro

An error has occurred during macro expansion. The most likely error is a missing closing bracket.

Macro too long (limit 100 chars)

A macro name is too long for the internal buffer.

Rules must be after target

make

maintain, update, and regenerate groups of programs (QNX)

A *makefile* syntax error, where a line beginning with a **<space>** or **<tab>** character has been encountered before a target line.

Too many options

The **make** utility has run out of allocated space while processing command-line options or a target list.

Too many rules defined for target

A target occurs multiple times, and each time has rules. A target may have only one set of rules.

Unexpected end of line seen

A target line without a colon has been encountered.

Environment variables:

MAKEFLAGS	This environment variable, if it exists, is read by the make utility, and is treated as a set of option characters to be used as the default options. Command-line options to make have precedence over these options. After make has started and the MAKEFLAGS variable has been read, all of the options except -d, -f, and -p are added to the MAKEFLAGS macro. The MAKEFLAGS macro is passed into the environment as an environment variable for all child processes.
SHELL	The value of the SHELL environment variable is always ignored. All other environment variables are used as macros as defined in the section “Macro Definition,” above.
WORKNODES	The list of nodes to be used for parallel rule execution, as defined in the section “Parallel Make,” above.

History: This implementation is based on original source written and placed in the public domain by Greg Yachuk. The source is used with permission from the author. He may be contacted at:

Informix Software Inc.
Menlo Park, CA 92025
greggy@informix.com | {uunet,pyramid}!infmtx!greggy
(415) 926-6300

See also: `cc`

Caveats: The `-n` option doesn't always correctly identify the targets that need to be made. The option never misses required targets, but sometimes includes targets that aren't required. It also incorrectly sets up the `$?` macro in these circumstances. This doesn't happen during full execution.

Target lines can't use the double colon (`: :`) syntax supported by some other **make** implementations.

Once a target is made, **make** assumes that the target file is present for the remainder of the run. If a rule subsequently removes that file and future targets depend on its existence, unexpected errors may result.

When searching for implicit rules, **make** sometimes uses several rules instead of a single rule. For example, the two rules `.c.o` and `.l.c` might be used, rather than the more direct `.l.o` rule.

If several command-line flags are run together, and contain either `-f` or `-d`, the whole set of flags is dropped from **MAKEFLAGS**.



Syntax: `mkdir [-p] [-m mode] dir...`

Options: `-m mode` After creating the directory, set the file permission bits of the new directory to the specified *mode* value.

The *mode* argument is a *symbolic_mode* string, as defined for the **chmod** utility. In the *symbolic_mode* strings, the *op* characters + and - are interpreted relative to the default file mode for that file type:

- + add permissions to the default mode
- delete permissions from the default mode
- = assign permissions

`-p` Create any missing intermediate pathname components.

dir A pathname at which a directory is to be created.

If you specify both the `-p` and `-m` options, any intermediate directories you have created have mode **u+wx**.

Examples: `mkdir /usr/debbie`

Description: The **mkdir** utility creates the directories specified by the *dir* operands, in the order the *dir* operands are specified.

The default file mode for directories is **a=rx** (777), with selected permissions removed in accordance with the file mode creation mask (see the **umask** utility).

For intermediate pathname components created by **mkdir**, the mode is the default modified by **u+wx** so that the subdirectories can always be created regardless of the file mode creation mask. If you want to assign different ultimate permissions for the intermediate directories, you can do so with the **chmod** utility.

When using `-p` with `-m`, each intermediate directory that doesn't exist is created with **u+wx** modes, regardless of the file mode creation mask. The specified *mode* applies only to the last directory specified. For example:

```
mkdir -p -m 777 dir/dir1/dir2
```


mkdir

make directories (POSIX)

would give **dir** and **dir1** the default permissions for intermediate directories (namely, **u+wx**). The directory **dir2** is given **a+rw** permission.



The default file creation mask influences the behavior of **mkdir**.

Exit status: 0 All the specified directories were created successfully, or the **-p** option was specified and all the specified directories now exist.

>0 An error occurred.

Asynchronous events: If the **mkdir** utility is terminated by a signal, some of the specified directories may have already been created.

See also: **chmod, rmdir, umask**

Syntax:	mkfifo [-p] [-m <i>mode</i>] <i>file</i> ...
Options:	-m <i>mode</i> After creating the FIFO special file, set the file permission bits of the new file to the specified <i>mode</i> value. The <i>mode</i> option argument is a <i>symbolic_mode</i> string, as defined for the chmod utility. In the <i>symbolic_mode</i> strings, the <i>op</i> characters + and - are interpreted relative to the default file mode for that file type, as follows: + add permissions to the default mode - delete permissions from the default mode = assign permissions -p Create directories in the FIFO's pathname, if required. <i>file</i> The pathname at which a FIFO special file is to be created.
Description:	The mkfifo utility creates the FIFO special files specified by the <i>file</i> operands in the order they're specified. The default file mode for FIFO files is a=rw (666) with selected permissions removed in accordance with the file mode creation mask. For intermediate pathname components created by mkfifo, the mode is the default modified by u+wx so that any subdirectories and the FIFO file can always be created regardless of the file mode creation mask. If you wish to assign different ultimate permissions for the intermediate directories, you can change these permissions afterward with the chmod utility.
Asynchronous events:	If the mkfifo utility is terminated by a signal, some of the specified FIFO special files or intermediate directories might have already been created, and may not be automatically removed.
Exit status:	0 All the specified FIFO special files were created successfully. >0 An error occurred.
See also:	chmod



Syntax	modem [<i>options</i>]							
Options:	-B <i>baud</i>	Assume this baud rate if the modem answers “connect” without a specified baud rate (default is 2400 baud).						
	-C	Don’t wait for CTS.						
	-D <i>file</i>	Log debug information. If file is a dash character (-), then output is sent to stderr .						
	-L	Used locked baud rate.						
	-R <i>minutes</i>	Resynchronize the modem by sending the initialization string (specified via option -i) every <i>minutes</i> minutes (default is every 15 minutes. You specify -R 0 to disable resynchronization).						
	-a <i>string</i>	Use <i>string</i> to answer the phone (default is “ ATA ”).						
	-b <i>baud</i>	Use this baud rate when programming the modem or receiving result codes (default is 2400 baud).						
	-c <i>command</i>	Start this command (default is login).						
	-g <i>file</i>	On connect, print the greeting message stored in <i>file</i> .						
	-i <i>string</i>	Use this string to initialize the modem (default is “ ATZ ”).						
	-p <i>parity</i>	Set the parity, number of data bits, and number of stop bits specified in <i>parity</i> . This consists of a 3-character string: <table> <tr> <td>parity</td> <td>data</td> <td>stop</td> </tr> <tr> <td>{e o m s n}</td> <td>{7 8}</td> <td>{1 2}</td> </tr> </table> The default is n81.	parity	data	stop	{ e o m s n }	{ 7 8 }	{ 1 2 }
parity	data	stop						
{ e o m s n }	{ 7 8 }	{ 1 2 }						
	-r <i>rings</i>	Wait for this number of rings before answering (default is 1). You specify -r 0 to suppress answering.						

modem

modem handler to answer calls (QNX)

-s seconds Delay this number of seconds before starting command (default is 1 second).

Examples: Note that you should start each of these commands via the **tinit** utility, as opposed to directly from the command line. For more information, see the *Administrator's Guide*.

modem A 2400 baud slow-speed modem.

modem -b 1200 -R 10 -D /etc/modem/log

A 1200 baud slow-speed modem used to capture debug output.

modem -g /etc/modem_msg -b 19200 -L

A high-speed modem that supports a locked baud rate and hardware flow control between itself and the computer. Many 2400 baud MNP modems, while not “high speed,” will typically operate in this mode with a 9600 baud data rate.

Description: The **modem** utility monitors a serial line connected to a Hayes-compatible modem. It is usually started by the **tinit** utility, in place of a login.



For testing, you can start **modem** on a serial line using I/O redirection in place of the **tinit** utility. This will operate **modem** on a one-shot basis. That is, when a call is answered by the **modem** utility, it will start a **login** (this can be changed with the **-c** option) and terminate. When the **login** user logs out, **modem** will not be restarted. For example:

modem -b 19200 -L -D - </dev/ser1 >/dev/ser1

The **-D** option will output to your screen all communication between your modem and the **modem** utility.

To configure a modem as desired, the **modem** utility can be told to send commands to the modem. It will then wait for the modem to issue messages concerning an incoming call. The modem can be

configured to auto answer or the **modem** utility can be told to command the modem such that **modem** answers the call itself.

After **modem** has answered the phone and determined the baud rate, it waits for the delay specified via the **-s** option. This delay avoids the burst of noise characters that often occur when two modems first connect. (On some modems, this has been observed to be up to 10 seconds.) After the delay, a greeting is displayed, provided the **-g** option was used to specify a greeting file. The **modem** program will then replace itself—via the *exec()* function call with either the **login** utility or the command specified with the **-c** option. The user may now log in as usual. If the user logs out, **tinit** restarts another **modem** program on the line and the whole process is repeated.

You can periodically reprogram a modem via the **-R** option. This is useful if the modem is powered off, then on again. This practice might also improve the reliability of some modems that “forget” their settings over time (this isn’t supposed to happen in a perfect world...).

The **-p** option sets the line parameters for the parity, number of data bits, and the number of stop bits. It consists of a three-character string. The following table summarizes the five most common configurations.

e71	even parity	7 bits	1 stop
o71	odd parity	7 bits	1 stop
m71	mark parity	7 bits	1 stop
s71	space parity	7 bits	1 stop
n81	no parity	8 bits	1 stop (the default)



To talk to another QNX system, **n81** should be selected.

The **-L** option should be used with high-speed modems ($\geq$ 9600 baud or 2400 baud MNP) that talk to the computer at a fixed baud rate and use hardware flow control to regulate the data. This option will enable QNX hardware flow control. Make sure you program your modem to support hardware flow control as well.

So that **modem** can detect the modem status (on or off line), you should configure your modem—if possible—to provide the CD (carrier detect) RS-232 signal to the computer. For a Hayes modem,

modem

modem handler to answer calls (QNX)

this command is typically “**AT&C1**”, but check the manual for your modem since this command does vary. When a loss of carrier occurs, a SIGHUP is sent to the session leader, typically the **login** shell. This will cause it to terminate, which will send a SIGHUP to non-background children. The net effect is to remove all the users’ processes, thus logging the user off.

So that **modem** and other programs can force the modem to hang up if necessary, the DTR (data terminal ready) RS-232 signal should also be connected from the computer to the modem. Your modem should be programmed to force a hangup and reset itself when the computer “drops” DTR. For a Hayes modem, this command is typically “**AT&D2**”, but check the manual for your modem since this command varies.

As part of the baud rate determination, **modem** posts an environment variable called **BAUD**, which is set equal to the baud rate for the modem connection. The **BAUD** variable is useful since many high-speed modems operate with the serial link between the modem and the computer at a fixed high speed (usually 9600 or greater), and then use hardware flow control to regulate the flow of data for slower connections. Since the true baud rate cannot be determined by the **stty** setting, **modem** posts this information as an environment variable so that the tasks started by the dial-up user can determine the true transfer rate.

To change the modem initialization string used by **modem**, you use the **-i** option. The default string is “**ATZ**”, but a suggested string is:

```
"AT E1 Q0 V1 X1 &C1 &D2"
```

where:

AT	start of <i>command</i>
E1	echo commands
Q0	result codes displayed
V1	English result codes
X1	display result codes
&C1	indicates carrier detected
&D2	DTR forces hangup

With a 2400 baud (or better) modem, these settings can usually be programmed into the non-volatile RAM in the modem with the **AT&W** command.



Don't place an **AT&W** command in an initialization string, since many modems support a limited number of writes to their non-volatile memory. You can program the non-volatile memory by hand when you first configure the modem via the **qtalk** utility.

If the modem settings are stored in the modem, you can usually use the default modem initialization string "**ATZ**", which causes the modem to program itself to the internally saved state. Also, the modem can often be programmed to reset itself to the internally saved settings on loss of carrier, or hangup forced by DTR driven low by the computer.

The modem initialization string supports a number of special characters that are processed by **modem** rather than being passed to the modem. These characters, and their actions, are:

- | carriage return
- ~ 1-second delay
- ˘ 100-msec delay
- ^ drop DTR for 1 second (forces a modem to hang up and reset)
- ! send a 500-msec break

As the modem initialization string is processed, any characters not in this special set are passed through to the modem, unmodified. Using these special characters, modem initialization strings that perform many steps can be easily constructed. For example:

```
"^~~~ATZ"
```

would force a hangup of the modem with DTR (with 3-second delay after the change of DTR) and then program it to reset to its saved state (check that your modem supports ATZ).

To accept incoming calls, your modem need not be programmed for auto-answer. The modem can be set to "not auto answer" (ATS0=0). The modem will emit "RING" statements when an incoming call is detected and **modem** will detect the "RING" and provide the answer

modem

modem handler to answer calls (QNX)

string (default is “**ATA**”) to force the modem to answer. This has the added advantage that your system won’t answer the phone unless it is powered up and the **modem** program is ready for the incoming call. We recommend operating your modem in this mode.

We also recommend you default your serial ports to 8 bits, no-parity, and 1-stop bit for communication with other QNX systems.

The **modem** utility will log connect messages to the file **/etc/acclog**, provided the file exists. For details, see the *Administrator’s Guide*.

Files: **/etc/acclog** (accounting records are appended to this file)

See also: **login, qcp, qtalk, tinit**

Syntax: **more** [-ceisu] [-n *number*] [-p *pattern*]
 [-/ *pattern*] [-t *tag*] [-x *tabstop*] [*file...*]

- Options:**
- c For each full screen of text that is displayed, clear the screen from the first line and display the next full screen of text.
 - e Stop after displaying the last line in the file. If the next command that displays text causes **more** to reach end-of-file again, **more** exits. If the file is shorter than a single screen, **more** exits at end-of-file regardless.
 - i Ignore case during searches. Uppercase and lowercase letters are considered identical.
 - n *number* Specify the number of lines that constitute a full screen of text. The *number* argument is a positive decimal integer. The -n option overrides any values obtained from the environment.
 - p *pattern* Search for a line that matches *pattern*. The current position is set to the first matched line. If no match is found, the first line in the file shall be the current position.
 - / *pattern* Same as -p *pattern*.
 - s Replace consecutive empty lines with a single empty line.
 - t *tag* Display the file containing the tag named by the *tag* argument. Note that for this to work, the file **tags** must reside in the current directory.
 - u Always display backspaces as control characters (e.g. as the two-character sequence **^H**) and leave **<carriage return><linefeed>** (**\r\n**) sequences alone.
- By default, **more** makes special use of backspaces and **<carriage return><linefeed>** (**\r\n**) sequences. If a backspace appears next to an underscore character, the character is displayed as underlined text, provided

more

display files on a page-by-page basis (POSIX)

the terminal type supports underlined text. If a backspace appears between two identical characters, the first character is displayed as bold text, provided the terminal type supports bold text display.

-x *tabstop* Set tabs at the positions specified by *tabstop*. The default is four spaces, unless the POSIX_STRICT environment variable is defined, in which case it is eight spaces.

file A pathname of an input file. If no *file* operands are specified, **more** uses the standard input. If a *file* operand is the dash character (-), the standard input is read at that point of the sequence.

Description: The **more** utility lets you view text files one screenfull at a time. The utility determines the number of lines that make a full screen by looking in the terminal database. However, the **LINES** environment variable can be used to override the value found in the database, and the **-n** option can be used to override the **LINES** variable.

If the standard output isn't a terminal device, the number of lines that make up a full screen of text is considered to be infinite. In a pipeline, all input files are copied to the standard output in their entirety. On terminals, **more** displays text one screen at a time.

The **more** command can be very useful when another utility prints more information to the standard output than can be displayed on a single screen. By piping the output to **more**, you can scroll through the displayed output at leisure. For example:

```
ls | more
```

pipes the output from the **ls** command to **more**, allowing you to scroll through the output.

**Environment
variables:**

EDITOR	Used to select an editor.
LINES	Interpreted as a decimal integer value to be used as the number of lines in a screenfull.
MORE	Interpreted as a string containing options described in the Options section of this utility, preceded by hyphens and separated by <blank> characters as on the command line. Command-line options override those specified in the MORE variable. The MORE variable takes precedence over the TERM and LINES variables.
TERM	Interpreted as the name of the terminal type.
POSIX_STRICT	Interpret options according to POSIX specifications.

Exit status:

0	Successful completion.
>0	An error occurred.

See also:

cat, ctags, less

10

11

12

Syntax: **mount** [-p *block_special_file*]
 [*block_special_file* *directory* [-r]] ...

Options: -p *block_special_file*

Read the partition table from the named block special file and mount all partitions found. The partitions will appear as block special files with the form:

block_special_file t *partition-type-num*

That is, if the block special file was named */dev/hd0* and the QNX 4.0 partition (type 77) was found, the block special file */dev/hd0t77* would be created.

block_special_file *directory* [-r]

Mount, as a filesystem named *directory*, the named block special file. If the -r option is given, mount it as a read-only filesystem.

Examples: Mount all partitions found on hard disk 0, and mount the QNX 4.0 partition as the root (this line is commonly seen in OS build files):

```
mount -p /dev/hd0 /dev/hd0t77 /
```

Mount floppy disk 0 as the directory */fd* and make it read-only:

```
mount /dev/fd0 /fd -r
```

Mount the ramdisk as the temporary directory:

```
mount /dev/ram /tmp
```

Mount the QNX 4.0 partition found on the second hard disk as the user directory:

```
mount /dev/hd1t77 /usr
```

Description: You use the **mount** utility to mount disk partitions as block special files, and to mount block special files as QNX 4.0 filesystems.

mount

mount partitions and filesystems (QNX)

Exit status: 0 Successful completion.
 >0 An error occurred.

See also: *Administrator's Guide:* **buildqnx**

Syntax: **mv** [-f|-i] [-V|v] *source_file target_file*

mv [-f|-i] [-V|v] *source_file... target_dir*

Options:

- f** Don't prompt for confirmation if the destination path exists. Overwrite even read-only files.
- i** Write a prompt to the standard error output before moving any file that would overwrite an existing file. If confirmation is received, overwrite the existing file. Otherwise, go on to the next file.

source_file The pathname of a file or directory to be moved.

target_file The new pathname of the file or directory to be moved.

target_dir The pathname of an existing directory that the source file is to move to.

QNX extensions: **-v** Be very verbose.

-v Be verbose.

Examples: **mv orange banana** In the current directory, rename the file **orange** to **banana**.

Description: **First syntax form**

mv [-f|-i] *source_file target_file*

The **mv** utility moves the file named by the *source_file* operand to the destination specified by the *target_file* operand. This first syntax form is assumed when the final operand you specify doesn't name an existing directory.

Second syntax form

mv [-f|-i] *source_file... target_dir*

For each *source_file*, the **mv** utility moves the file to a destination file in the directory named by the *target_dir* operand. The destination's filename under the target directory will be the same as its basename (final path component).

mv

move files (POSIX)

For example:

```
mv dir/dir/myfile /existingdir
```

will move `dir/dir/myfile` to `existingdir/myfile`.

This second syntax form is assumed when either the destination names an existing directory, or when more than one source file is specified.



By default, **mv** overwrites an existing file without warning or confirmation whenever you have write permission on the file.

The **mv** utility asks you for confirmation if the following conditions are met:

- you haven't specified the **-f** option
- you lack write permission
- the standard input is a terminal

Upon receiving confirmation, **mv** overwrites the target file. It can do this only if the user owns the file or is a superuser.

If you want **mv** to request confirmation before overwriting any file, specify the **-i** (interactive) option. If you want **mv** to overwrite whenever possible without asking for confirmation, specify the **-f** (force) option.

Exit status:

0	All input files were moved successfully.
>0	An error occurred.

Consequences of errors:

If the copying of a directory is prematurely terminated by a signal or error, **mv** may leave a partial copy of the directory at the destination. In this case, the directory tree at *source_file* will not have been modified.

Files: As long as the input files specified by each *source_file* are on the same device as the target, the *source_file* operand can be of any file type. If the source and target reside on different devices, the *source_file* will be copied to the target and then removed. If the *source_file* is a directory, this means that any FIFO or character special files under the original directory will not be copied. Since the copy will not be 100% successful, the original *source_file* will not be removed.

See also: `cp`, `rm`



Syntax: `nameloc [-e node] [-p seconds] [-s node] [-t seconds]`

Options:

- `-e node` End node to poll (default is number of nodes).
- `-p seconds` Poll period (default is 1 second).
- `-s node` Start node to poll (default is 1).
- `-t seconds` Timeout on slow/hung nodes (default is 5 seconds).

Examples:

```
nameloc &
nameloc -p 5 &
nameloc -s 1 -e 9 &
```

Description: The **nameloc** utility runs in the background and provides a network-wide naming service for all processes running under the operating system. You may run **nameloc** on up to 10 nodes in the network. Running multiple copies provides redundancy in case a node running **nameloc** fails. Three copies on three different nodes is usually more than sufficient.

A process may register one or more symbolic names. These names may be local to the node on which the process is running or global to the entire network. Global names all start with a slash. For example:

<code>qnx/fsys</code>	local name
<code>company/xyz</code>	local name
<code>/company/xyz</code>	global name

The convention of prefixing a name with a company or product name is encouraged to reduce name conflicts between different vendors. The name locator is NOT required to locate names on the same node as the process performing the lookup. It is required if you wish to locate global names on other nodes.

Upon starting, **nameloc** will immediately poll each node in the network for its list of global names. It will then go into a slow poll mode to refresh this information. This slow poll period may be changed using the `-p` option. The slow polling is not the major means by which **nameloc** keeps informed of process global names in the network. Each time a name is registered or removed, all name locators are immediately informed. The slow poll is a safety net to

nameloc

name locator (QNX)

handle extraordinary error conditions which might cause it to miss an attach or detach request.

When polled by a name locator, each node in the network remembers the polling node. It will remember up to the last 10 different nodes it was polled from. If it fails to receive a poll after a 10-minute period, it will forget that node and thus that name locator.

See also: **sin**

Syntax: **netinfo** *node*

node The node for which events are displayed.

Description: The **netinfo** utility asks the network manager for the contents of its circular buffer of significant network runtime events. For each of these events, **netinfo** displays a line of output that consists of:

- the date and time of the network event
- the node number associated with the event
- the event’s numerical code
- English text that explains what the error signifies

The network events usually consist of the following:

- reconfigurations caused by nodes joining and leaving the network
- unidentified received network packets, caused by the QNX 2 poller on the same physical Arcnet cable attempting to talk to the QNX 4 node.

At the end of the list of network events, you’ll also see:

- the starting physical memory segment of the network card
- the hardware interrupt used by the network card
- the current values of the 9026 Arcnet chip’s status register
- the “best guess” of the network’s state—either the network card is or isn’t seeing a token being passed

Normally Transmit Acknowledge (TMA) and Transmitter Available (TA) are set to a logic one, and Receiver Inhibited (RI) and Reconfiguration (RECON) are set to a logic zero. The other status bits usually never change: ETS2 and ETS1 should always be set on, POR and TEST should always be set off. Because the RECON interrupt isn’t enabled, RECON can lag behind the actual state of the network—RECON will be updated and recorded upon the next packet received or transmitted on the network.

netinfo

display network events (QNX)

The “best guess” of the network’s state can lag behind the actual state of the network—it will be updated the next time a network transmission is attempted if in fact the token is not being passed.

Exit status: 0 Successful.
 >0 An error occurred.

See also: **sin**

Syntax: **newgrp** [-1] [*group*]

Deprecated version:

newgrp [-] [*group*]

Options: -1 (*or deprecated version: -*)

(the letter “e”) Change the environment to what would be expected if the user actually logged in again.

group A group ID name.

Description: The **newgrp** utility changes the current process’s real and effective group ID. The user remains logged in and the current directory doesn’t change, but calculations of access permissions to files are performed on the new real and effective group IDs.

If no operands are specified, **newgrp** changes the group ID back to the group identified in the invoking user’s entry in the user database.

If you don’t use the -1 option, the environment variables remain unchanged.

Note that group passwords aren’t supported in QNX. Only the superuser (user ID 0) may change the group ID to a group that the user doesn’t belong to.

Files: /etc/group
 /etc/passwd

See also: su



Syntax: `onnode nodenum command`

nodenum The number of the remote node that will run the specified command.

command The process that will be run on the specified node.

Examples: The following command:

```
onnode 1 sin
```

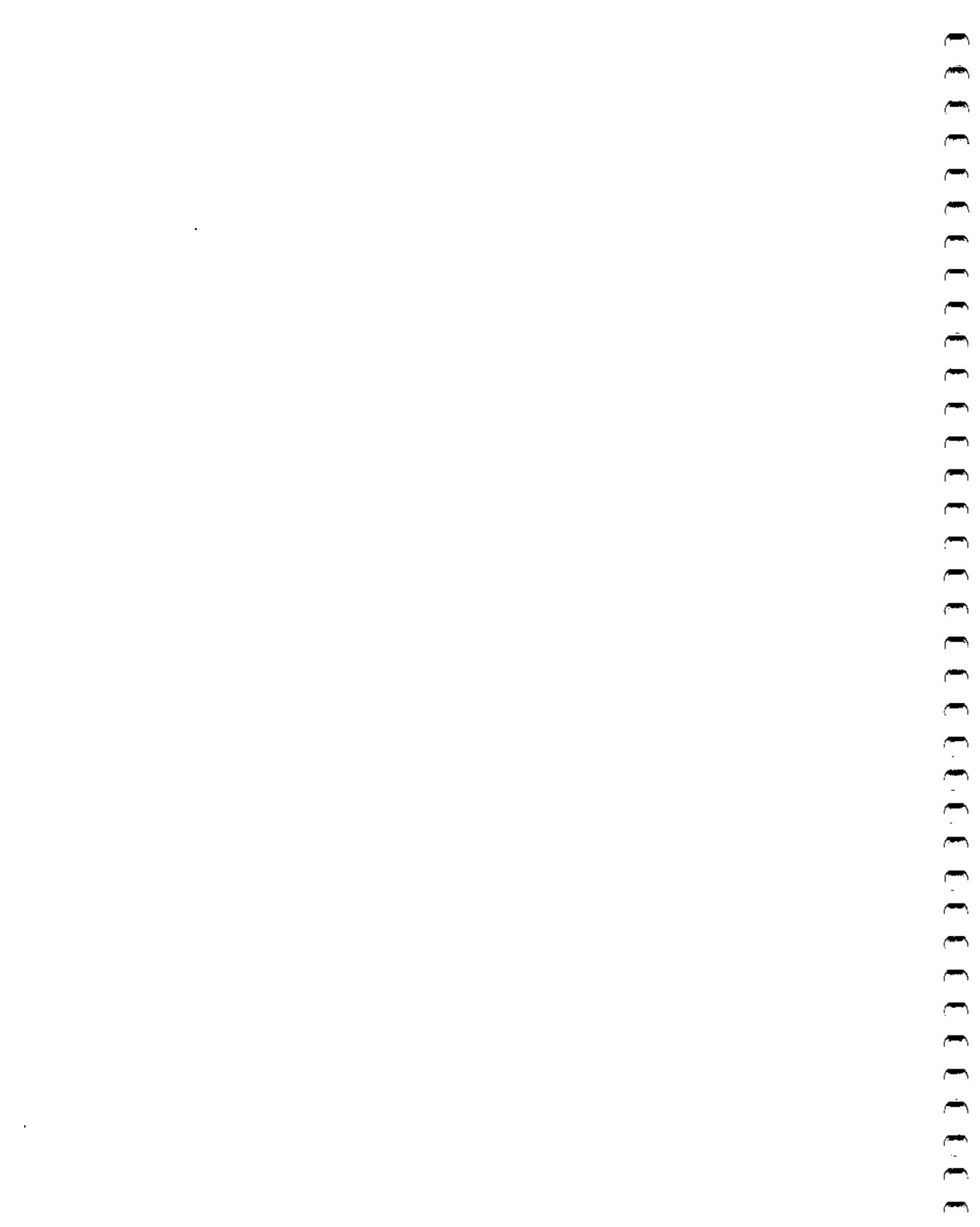
is analogous to the syntax:

```
//1 sin
```

Description: The **onnode** command is a Shell built-in that executes the specified process on the indicated node.

Exit status: The **onnode** utility exits with the value of the specified command.

See also: **ontty, sh**



Syntax: **ontty** *device command*

device The name of the device that will run the specified command.

command The process that will be run on the specified device.

Description: The **ontty** command is a Shell built-in that executes the specified command on the indicated device. The command is detached from your terminal.

If this command is to be run on a remote node, then the remote node number may either precede **ontty** or it may be included as part of the device name. For example, both of the following commands invoke a process called **monitor** on **/dev/ser1** of node 4:

onnode 4 ontty /dev/ser1 monitor

OR

ontty //4/dev/ser1 monitor

The **ontty** command operates by redirecting the program's **stdin**, **stdout**, and **stderr** to the indicated device. If the device is remote, the **ontty** command remotely executes the program on that node.

See also: **onnode, sh**



Syntax: `passwd [name]`

Description: The **passwd** utility allows a login password to be changed or a new username to be created.

If you are changing your password, **passwd** prompts for the old password and then for the new password. The new password must be entered twice, to avoid typing mistakes. Only the owner or the superuser may change a password.

The superuser may use this utility to create a new user account by entering:

`passwd new_user_name`

When creating a new user account, **passwd** will prompt for information, such as the user's group list, home directory, and shell.

Files: `/etc/passwd`

See also: `login, su`



Syntax: `pr [+page] [-column] [-a] [-d] [-e [char] [gap]] [-f] [-h header] [-i[char][gap]] [-l lines] [-m] [-n[char][width]] [-o offset] [-r] [-s[char] [-t] [-w width] [file...]`

Options:

- `+page` Begin printing at the page number of the formatted input that is specified by *page*.
- `-column` Print output in columns that is *column* spaces wide and is in text column rank order (default number of columns is 1). When you specify this option, options `-e` and `-i` are assumed for multiple text-column output. If you specify the `-t` option, `pr` uses the minimum number of lines to display the output.

The `-m` option is not allowed when the `-column` option is used.
- `-a` Print multiple-column output across the page. You should use this option only with the `-column` option.
- `-d` Print double-spaced output; append an extra `<newline>` after every `<newline>` found in the input.
- `-e [char] [gap]`
Expand input `<tab>`s to column positions *gap*+1, 2**gap*+1, 3**gap*+1, etc. If *gap* is zero or is omitted, `pr` assumes default `<tab>` settings at every eighth position. All `<tab>` characters in the input are expanded into the appropriate number of `<space>` characters. If you specify any non-digit character, *char*, the character is treated as the tab expansion character.
- `-f` Use a `<form-feed>` character for new pages (default is to use a sequence of `<newline>` characters). Prior to displaying the first page of output, an `<alert>` character is written to standard output.
- `-h header` Use the string *header* for the header, instead of the filename.

-i [*char*] [*gap*]

In output, replace multiple **<space>** characters with **<tab>**s wherever two or more adjacent **<space>** characters reach column positions $gap+1$, $2*gap+1$, $3*gap+1$, etc. If *gap* is zero or is omitted, **pr** assumes default tab settings at every eighth position. If you specify *char* (any non-digit character), the character is used as the output **<tab>** character.

-l *lines* (the lowercase letter “e”) Set the page length to *lines* (default is 66 lines). If *lines* is smaller than the number of lines required for the header and trailer, **pr** suppresses both the header and the trailer, as if the **-t** option were in effect.

-m Merge files. Standard output is formatted so that **pr** writes one line from each specified file side by side into text columns of equal fixed widths, in terms of the number of column positions.

-n [*char*] [*width*]

In output, provide line numbering that is *width* digits wide (default width for line numbering is 5 digits). The number occupies the first $width+1$ column positions of each text column or each line of **-m** output. If *char* (any non-digit character) is given, the character is appended to the line number to separate it from whatever follows (default for *char* is a **<tab>** character). The maximum width you may specify is 9.

-o *offset* Precede each line of the output by the number of **<space>** characters specified by *offset*. (default number of spaces is zero). The number of column positions per line is the sum of the width (from the **-n** option) and the offset.

-r Don't print diagnostic reports on failure to open files.

-s [*char*] Separate text columns by the single character *char* (default for *char* is the **<tab>** character).

- t** Print neither the identifying header nor the trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.
- w *width*** Set the width of the line to *width* column positions for multiple text-column output only (default is 71).
- file*** The pathname of a file to be printed. If you don't specify any files, or if *file* is the dash character (-), the standard input is used.

The options **-l** (the lowercase letter “el”), **-o**, **-s**, and **-w** don't allow the option letter to be separated from its argument. The options **-e**, **-i**, and **-n** require that both arguments, if present, not be separated from the option letter.

Description:

The **pr** utility is a printing and pagination filter that prints files to the standard output.

By default, the listing is separated into pages, each with a header that includes the page number, date, time, and the file's pathname. The header is of the form:

```
\n
\n
MMM DD HH:MM YYYY Page N \n
\n
\n
```

The trailer is simply the 5 **<newline>** characters required to advance to the next top of form (unless **-f** is used, in which case a **<form-feed>** will be sent). Text columns are of equal width; there is at least one **<blank>** separation character between text columns. Lines that don't fit into a text column are truncated. If standard output is associated with a terminal, diagnostic messages are suppressed until **pr** has completed processing.

Exit status:

- 0 All files were written successfully.
- >0 An error occurred.

pr

print files (POSIX)

Environment variables:	TZ	Specifies the local time zone for use in writing header lines.
Asynchronous events:	If pr receives an interrupt while printing to a terminal, it flushes all accumulated error messages to the screen before terminating.	
See also:	lp	

Syntax: `prefix [options]`

Options:

- `-A prefix=string` Add system prefix with alias *string*.
- `-D prefix` Delete system prefix.
- `-R prefix=string` Replace system prefix with new alias *string*.
- `-a prefix=string` Add local prefix with alias *string*.
- `-d prefix` Delete local prefix.
- `-n node_id` Print the prefix string from the indicated node.
- `-r prefix=string` Replace local prefix with new alias *string*.

Examples:

```
prefix
prefix -n 20
prefix -A /usr1=//10/usr1 -A /usr2=//14/usr2
prefix -A /dev/printer=//12/dev/spool
prefix -a /=//10
prefix -r /=//20
prefix -D /dev/printer
```

Description: The pathname space is administered through the use of two forms of prefixes:

- prefixes registered by I/O managers
- aliases

The **prefix** utility allows you to examine the full contents of the pathname prefix tree and to modify the alias portions of it.

A prefix tree is essentially an in-memory pathname redirector that is maintained on each computer. When a file is opened, its pathname is matched against the prefix strings in order to direct the request to open the file to the appropriate I/O manager.

For example, the character device manager **Dev** registers the **/dev** prefix. Therefore, when a process calls *open()* with **/dev/xxx**, a prefix match of **/dev** will occur and the *open()* will be directed to **Dev** (the prefix's owner).

prefix

examine or change the pathname prefix tree (QNX)

The form of an I/O manager prefix is as follows:

prefix=pid, unit

where *pid* is the process ID of the I/O resource manager and the *unit* number is a single character assigned by the I/O manager. For example, if **Fsys** is process 3 and **Dev** is process 5, then the prefix tree might appear as:

/=003, 2

/dev=005, 0

/dev/hd0=003, 0

/dev/hd0t77=003, 1

Note that **Fsys** has registered three prefixes:

- the root (/)
- a block special file representing the entire hard disk (**/dev/hd0**)
- a block special file representing the partition containing a QNX 4.0 partition (**/dev/hd0t77**)

In cases where prefixes overlap, the longest match always wins. Therefore, in the example above, references to **/dev/hd0** and **/dev/hd0t77** are directed to **Fsys** rather than to **Dev**, while all other pathnames starting with **/dev** are still forwarded to **Dev**.

Aliases

In addition to this absolute redirection to an I/O manager, another form of prefix, called an *alias*, lets you replace a matched prefix with a new prefix, which is then re-applied to the prefix tree. Unlike the I/O manager form of prefix, aliases can be modified using the **prefix** utility. An alias is of the form:

prefix=replacement_string

Aliases are often used to map remote resources into the local pathname space. For example, a diskless workstation might typically create an alias for its root (/) to a filesystem on another node:

prefix -A /=//10/

Aliases can also be used to create special device names. For example, if a print spooler is running on node 20, you could alias a local printer pathname to it as follows:

```
prefix -A /dev/printer=//20/dev/spool
```

System and private prefix trees

Every process has two prefix trees associated with it:

- a system prefix tree, which is common to all processes
- a private prefix tree, which is private to the process and any children it subsequently creates

A pathname is always applied to both prefix trees. In the event that a match occurs in both trees, the longest match wins. In the case of a tie, the private prefix wins.

Since the private prefix tree takes precedence over the system prefix tree, and since it is inherited, it provides an ideal mechanism to ensure that child processes created on remote nodes resolve their pathnames in a manner consistent with the parent.

The command-line options in capital letters, **-ADR**, modify the system prefix tree while the corresponding options in lowercase letters, **-adr**, modify the private prefix tree. Changing the system prefix tree affects all running processes; changing the private prefix tree affects only the current process and any children it subsequently creates.

The **-a** and **-A** options add a new prefix to the appropriate prefix tree. The prefix being added must not already be in the tree.

The **-r** and **-R** options delete, then add, a prefix to the appropriate prefix tree.

The **-d** and **-D** options delete an existing prefix from the appropriate prefix tree.

See also: *User's Guide*

Caveats: Since redefining the system prefix aliases affects all the processes on the computer, you should do this with caution, especially when dealing with definitions for root (/).



Syntax: **ps** [-A|-a|-d] [-g *grplist*] [-p *proclist*]
 [-t *termlist*] [-u *uidlist*]

Options:

-A	Print information for all accessible processes.
-a	Print information for all processes associated with terminals except for process group leaders.
-d	Print information for all processes except group leaders.
-g <i>grplist</i>	Print information only for processes whose group ID numbers are given in <i>grplist</i> .
-p <i>proclist</i>	Print information only for processes whose process ID numbers are given in <i>proclist</i> .
-t <i>termlist</i>	Print information only for processes associated with terminals given in <i>termlist</i> . Terminal identifiers are given in the form of the session ID (a small positive integer).
-u <i>uidlist</i>	Print information only for processes whose user ID numbers are given in <i>uidlist</i> .

Description: The **ps** utility prints information about processes, subject to having the appropriate privilege to obtain information about those processes.

If you specify no options, **ps** prints information about processes associated with the current terminal. The output includes the process ID, terminal name, cumulative execution time, and command name of each process.

The options that use lists (**-g *grplist***, **-p *proclist***, **-t *termlist***, **-u *uidlist***) can have the list specified in one of two forms:

- a list of identifiers separated from one another by commas
- a list of identifiers enclosed in double quotes and separated from one another by commas and one or more **<blank>** characters, or a combination of commas and **<blank>** characters.

A list is a single command-line argument.

ps

report process status (POSIX)

If you specify any of the **-g** *grplist*, **-p** *proclist*, **-t** *termlist*, or **-u** *uidlist* options, the initial set of processes selected by **-a**, **-A**, or **-d** is intersected with the processes selected by these options. If none of **-a**, **-A**, or **-d** is specified, **-A** is used as the initial set of processes.

Exit status: 0 Successful completion.
 >0 An error occurred.

See also: **sin**

Syntax: **pwd**

Description: The **pwd** utility writes the fully qualified pathname of the current working directory to the standard output. When displayed, the current working directory is preceded by a node number. For example, if Debbie issues the **pwd** command from her **debbie** directory on node 7, the following is printed:

```
//7/usr/debbie
```

Exit status: 0 Successful completion.
 >0 An error occurred.

See also: **cd, ls**



Syntax: Send files:

```
qcp [device] se [options] src_file[, dst_file] ...
    [x=index_file]...
```

Receive files:

```
qcp [device] re [options] [-f filename|-p prefix]
```

Options:

<i>device</i>	Name of QNX device to use. The default is the device connected to stdin and stdout .
-v	Be verbose. Display error status while transferring files.
-f filename	Force received files to have this name.
-l logfile	Append the file transfer event to the logfile.
-m	Suppress making directories for received files.
-n	Receive only files that are newer than existing files.
-p prefix	Place this path prefix on the names of any received files.
-q	Be quiet. Display nothing during the transfer.
-r	Relax timing. Double timeouts and quadruple retry counts.
-s packet_size	The size of transmitted data bursts (default is 2048).
-t	Display today's date on received files.
x=index_file	A list of filenames; qcp will send each file named in the list. Multiple x=index_file options can be specified and intermixed with singly named files to transmit.

Description:

The **qcp** utility provides an error-checked file transfer protocol that **qtalk** uses to transmit or receive files. This protocol is both highly efficient on packet switched networks and highly reliable through the use of 16-bit CRCs (Cyclic Redundancy Check).

qcp

Quantum communications protocol (QNX)

The **qcp** utility automatically sends files with their pathnames, attributes, permissions, and date fields intact. The **qcp** protocol is ideally suited for use over public packet switch networks (X.25), as well as direct modem-to-modem connections. If communication errors are encountered, portions of the file will automatically be re-sent until the far end acknowledges correct reception of the file.

If you're using **qtalk** to communicate with a remote system, you can both send and receive files to the remote system with the **qcp** utility. To cause a file to be sent from the remote system to your local system, type a command of the following form into the **qtalk** session you have connected to the remote system:

```
qcp se file1 file2,file3 x=file4
```

This will cause the remote end to send the files to you; **qtalk** will automatically start **qcp** to receive the file. The files sent will be:

- **file1**
- **file2** (**file2** is received with the filename **file3**)
- all the files named in the index file **file4**



You can create index files easily with the **ls -p** command.

To send a file to the remote system, type the following command into the remote shell:

```
qcp re
```

then use the <Ctrl>-<A> <Ctrl>-<S> key sequence to cause **qtalk** to send a file to the remote **qcp** utility.

If you want **qcp** to send a file through an explicitly named device, use a command of the form:

```
qcp /dev/ser1 se filename
```

where **/dev/ser1** is the port to send through. To receive from a particular device, use a command similar to:

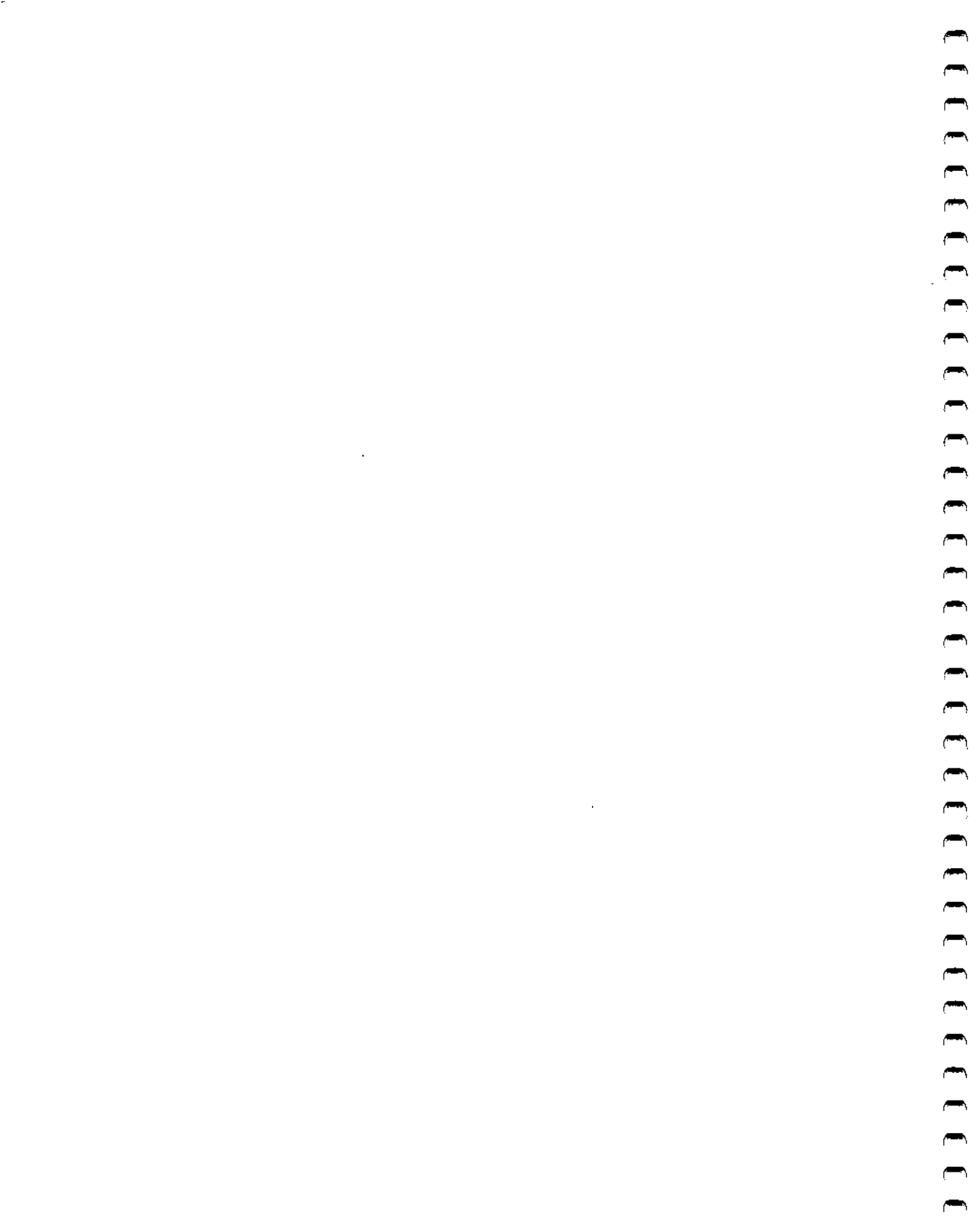
```
qcp /dev/ser1 re
```

You can abort a **qcp** file transfer in progress by pressing either <Esc> or <spacebar>. In turn, **qcp** displays a prompt asking for confirmation of the action. To confirm aborting **qcp**, you enter **y**. If a remote **qcp** in a receive state must be shut down, the following control-character sequence aborts it:

^V ^X ^X

Note that **qcp** will automatically remove any partially transferred files.

See also: **ls, qtalk**



Syntax: `qtalk [options] [system]`

Options:

- `-P` Ignore 8th bit of received characters.
- `-c hh` Specify character that invokes **qtalk** commands (default is **^A**).
- `-d hh` Replace ASCII rubout with this character.
- `-e` Enable local echo.
- `-i` Support flow control on the incoming data.
- `-k hh` Kick character.
- `-l logfile` Log a recording of the **qtalk** session in *logfile*.
- `-m modem` Name of async device (default is `/dev/ser1`).
- `-o options` Option(s) to be used for file transfer protocol.
- `-p hh` Pause or turnaround character.
- system* The name of the system you want **qtalk** to call.

Examples:

- `qtalk home` Phone home.
- `qtalk -e -d 08` Communicate with a machine that doesn't echo (half duplex) and expects an ASCII **<backspace>** (08 hex) to delete characters.
- `qtalk -l /dev/par` Communicate with another system and print hardcopy record.
- `qtalk -k 11 -p 0d` Prepare to send a file to a computer that sends **^Q** when ready for input.
- `qtalk -m //2/dev/ser2` Use the `/dev/ser2` serial port on node 2.

qtalk

talk over communications line (QNX)

Description: The **qtalk** utility lets QNX users communicate with other computers via a serial line which is usually connected to a modem. The destination may be another host computer in which case **qtalk** allows your computer to be used as a terminal. The **qtalk** utility also lets two QNX users communicate and transfer files.

The **qtalk** utility sends any characters you type on the keyboard to the other system via the modem. Any characters received by the modem are displayed. In local echo mode, typed characters are echoed on the display as well as being sent over the modem.

You can log a recording of a **qtalk** session with the **-l** option:

```
qtalk -l /dev/par
```

OR

```
qtalk -l /tmp/logfile
```

You can use a special command character to set special modes and options while you're within the **qtalk** environment. This special character defaults to Control A (^A) unless you change it with the **-c** option when you invoke the **qtalk** utility. The character you enter after this special command character is interpreted by **qtalk** to perform some special function. If you enter <Ctrl>-<A> twice, a single ^A is echoed to the modem.

Commands **b** (break)

Send a break over the modem. You may also send out breaks by pressing the break key (<Ctrl>-<Break> on the console keyboard).

c (change directory)

If you specify the **c** command, **qtalk** prompts you for a new directory name, and will attempt to move to that directory. If **qtalk** moves successfully to the directory you specify, the directory becomes the current working directory (see the **pwd** utility) for the duration of **qtalk**, or until you change directory again. When **qtalk** terminates, your current working directory reverts to the directory you were in when **qtalk** was invoked.

d (dial system)

If you specify the **d** command, **qtalk** prompts you to enter a system name. If that name is found in the **dial.dir** file, the associated string (usually a modem dialing command) is sent to the modem. If you enter a question mark (?) for the system name, the contents of the dialing directory are displayed, and you are prompted again for the system name. To abort the dialing command and return to normal communications mode, press <Enter> without entering a system name.

Dialing is implemented by looking up the system name (which can also be specified on the command line when **qtalk** is invoked) in the **/etc/config/dial.dir** file. This file is of the form:

```
system-name      dialing-command
```

The system name may be any string of characters (except for spaces and tabs) of any length. The dialing command consists of any characters (including spaces and tabs) that are sent to the modem. Typically, this would be the dialing command for your modem, and the phone number to dial. The system name and the dialing command are separated by one or more spaces or tabs, or a combination of both.

This is an example of a dialing directory entry that instructs a Hayes-compatible modem to call Quantum's QUICS update system:

```
quics ATD16135910934
```

e (echo)

If you specify the **e** command, the local echo feature is toggled. Some systems expect the "terminal" to perform local echoing.

h (hang up)

If you specify the **h** command, the CTS/RTS lines are lowered for approximately ½ a second. This permits modems that support hardware hangup to do so.

l (log)

Begin or end logging of this session. If no log file is open, **qtalk** asks for the name of a file to log into. If logging is already in progress, it is terminated and the log file is closed. The **l** command records every character that is sent or received in the log file.

o (transfer options)

If you specify the **o** command, **qtalk** prompts for the options to use when invoking the built-in file transfer protocol (**qcp**). The options are used for both sending and receiving files. Often, the **qcp -n** (“newer”) option is used to condition the transfer to ignore files which are older than the copy you already have on disk. This can be a real time-saver when downloading an update from QUICS.

For example:

Transfer options: -n

p (parity)

Ignore parity (top bit) of received characters. If this option is already set, turn it off.

q (quit and hang up)

This executes a hangup command (see **h** above) before exiting from **qtalk**.

r (invoke a receive)

Invoke the built-in file transfer protocol (**qcp**) to receive a file. Note that **qtalk** will automatically invoke **qcp** if it recognizes a received **qcp** startup sequence.

s (send a file)

Send a file using the built-in file transfer protocol (**qcp**). This sends a file to another system running the same protocol, which is far more secure than simply writing the file to the modem.

The **qcp** protocol is a secure, high-speed protocol specifically designed for QNX. It permits information about the file (the file date, attributes and permissions) to be transferred, as well as the file contents. The **qcp** protocol is ideally suited for use over public packet switch networks (X.25), as well as direct modem-to-modem connections. If communication errors are encountered, portions of the file will automatically be re-sent until the far end acknowledges correct receipt of the file.

You can send more than one file by specifying the **x=index_file** option when **qtalk** asks for the file to send. This file will contain a list of files to send, one per line. You can also specify more than one filename, separated by spaces.

The **qtalk** utility lets you follow the name of the file to send with the name of the destination file. You separate the two filenames with a comma. This is also true for filenames within an index file (option **x=**). For example:

```
Send file(s)?      file1 main.c,new_main.c
```

sends the file **file1** as **file1** and the file **main.c** with the name **new_main.c**. If you don't specify a new name, **qtalk** creates a file with the same name as the file that is sent.

Files received by **qtalk** using the **qcp** protocol have the same attributes and date as the file on the sending machine.

Protocol transfers require that the modem port be configured for 8-bit data (see the **stty** command).

u (upload text)

Upload text to another system. This command permits text files to be sent to another computer using a simple handshaking protocol based on the pause and kick characters, which you specify on the command line. If local echo isn't enabled (see the **e** command, above), **qtalk**

waits for the character to be echoed by the remote system before sending the next character. If local echo is enabled, the text is written to the modem with pauses only when the pause character is encountered.

You may abort a text upload at any time by pressing any key.



You must specify both the pause and kick characters to use the upload handshaking protocol. If neither character is defined, the result is the same as issuing a **w** (write) command (see below).

w (write)

Write a file to the modem. The file is transmitted over the modem, and also echoed on the display if local echo is enabled. The **write** command also allows files to be transferred to hosts or other users with handshaking, but with no error checking.

You may abort writing to the modem at any time by pressing any key on.

The primary difference between **upload** and **write** is whether or not to wait for the character to be echoed by the remote system before sending the next character.

x (exit)

Exit from **qtalk** without performing a hangup. It is good practice to get in the habit of using the **q** (quit and hang up) command for leaving **qtalk**, unless you really don't want to perform a hangup.

! (execute a shell command)

This command lets you execute any command from within **qtalk**. You'll probably find that you'll use the **!** command often to:

- execute the **ls** command in order to see what files are in your current directory prior to uploading or sending some of them
- execute non-native file transfer protocols (such as Xmodem or Kermit) from within **qtalk**

Using very high-speed modems, or producing a log on slow printers or floppy disks, may cause some characters to be lost. You can enable input flow control prior to invoking **qtalk** (see **stty**), or by specifying the **-i** option when you invoke **qtalk**. Either method prevents characters from being lost in these cases, provided that the machine sending the data supports flow control of its output.

The **qtalk** utility lets you transfer files through either of two methods. You invoke the simpler method with the **upload**, **write**, and **log** commands to send and receive text files to or from another system. No error checking is performed, however, so you should use this method only when communication lines are good (i.e. direct connect lines or local modem connections).

The second method, involving **send** and **receive** commands, uses a more sophisticated protocol that includes error checking and re-transmission when transferring files to other systems.

To select the protocol that is used by the **upload** command, you can specify two parameters on the command line:

- p=hh** Pause or turnaround character (in hex). If **upload** detects this character in the file it is sending, **qtalk** pauses, waiting for a kick character to be received. If you set pause to a CR (0D hex), **upload** sends one line at a time.
- k=hh** Kick character (in hex). This character is used in conjunction with the pause character by **upload** to handshake with other computers when transferring files.

These parameters allow the protocol to be adjusted according to the characteristics of the hardware at each end.

To transfer files to other systems that can accept input without needing to pause for any form of handshaking, you can use the **write** command. For slower systems—especially systems running full-duplex—the **upload** command is more appropriate. You can use the **log** command to take “snapshots” of data from a host computer at slow speed.

qtalk

talk over communications line (QNX)

To send a file to a host, you can use **qtalk** to enter into the input mode of the host's editor. You can then use the **upload** command or the **write** command to send a QNX file to the host. You should invoke **qtalk** with the **-p 0d** commands so that one line at a time is sent. Kick should be the prompt character that is printed by the host's editor whenever it's ready to receive a new line. On some systems, this might be a **^Q**, or it may be a prompt character such as a dot (**.**).

You'll find the **-d delchar** option to be very useful when you're communicating with computers that have a rubout character that's different from the one you're used to. Many systems use the <Backspace> key (08 hex) to erase a character. QNX systems default to the ASCII rubout character (7F hex). If you type:

```
qtalk -d 08
```

qtalk translates your <Rubout> key into backspace automatically.

See also: **qcp, stty**

Syntax: `rm [-Rfir] [-d] [-l n] [-v] file...`

Options: `-R` Recursively remove files and subdirectories under directories given as arguments. This process removes the directory and the entire file tree under it.



The `-R` option must be used with caution, as it removes directories, subdirectories and files. Using the `-i` option with `-R` adds a measure of safety, however, by turning on interactive prompting before each file or directory is removed.

`-f` Force each specified file to be removed without prompting for confirmation.

`-i` Be interactive; request confirmation before removing each existing file.

`-r` Equivalent to the `-R` option.

file The pathname of a file to be removed.

QNX extensions: `-d` If the `-R` option is specified, remove the files but leave the directory tree intact (i.e. no `rmdir` is performed).

`-l n` (the lowercase letter “el”) If the `-R` option is specified, recurse only *n* levels down a directory tree.

`-v` Be verbose. Print out files and directories as they are removed.

Examples: `rm a.out core` Remove the `a.out` and `core` files.

`rm -Rf junk` Remove the directory `junk` and all its contents, without prompting.

Description: The `rm` utility removes each specified file from a directory.

By default, `rm` refuses to remove any file that names a directory. This may be overridden with the `-R` or `-r` options. In any case, `rm` always refuses to remove the current working directory.

If a *file* operand has been specified but doesn’t exist and the `-f` option hasn’t been given, a message is written to the standard error

rm

remove files (POSIX)

output. If **-f** has been given, the error message isn't written. In either case, **rm** goes on to any remaining *files*.

The **rm** utility doesn't necessarily remove the file itself. A file may have more than one link; that is, it may be known by more than one name in the filesystem (see the **ln** utility for information on creating links). The **rm** utility breaks one such link; it dissociates the file from one name. If this is the only link, the file data becomes inaccessible and the file space is returned to the system for reuse. Otherwise, the data remains accessible via other names.

Exit status: 0 All the named files were removed.
 >0 An error occurred.

See also: **rmdir**

Syntax: `rmdir [-p] dir...`

Options: `-p` If the parent directory of a child directory becomes empty when the child is removed, remove the parent as well. Recurse up the directory tree where possible.

dir The pathname of an empty directory to be removed.

Examples: Remove the subdirectory `oldfiles` from the `usr/fred` directory:

```
rmdir usr/fred/oldfiles
```

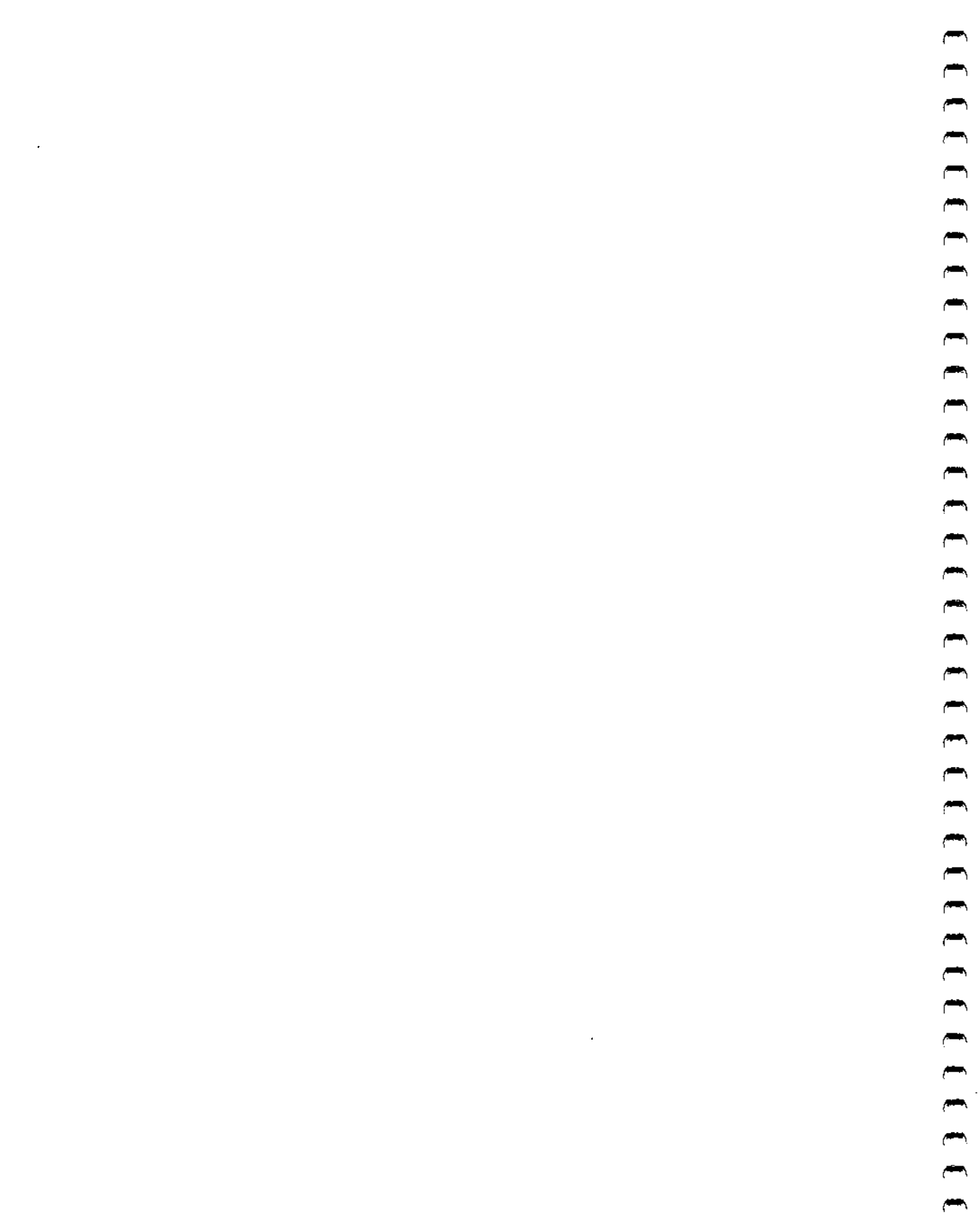
Description: The `rmdir` utility removes the directory specified by each *dir* operand, provided the directory is empty.

The `rmdir` utility processes directories in the order they're specified on the command line. If you specify a parent directory, you should specify its subdirectory *before* the parent directory. That way, the parent directory will be empty when the `rmdir` utility tries to remove it.

Exit status: 0 Each directory specified by a *dir* operand referred to an empty directory and was removed successfully.

>0 An error occurred.

See also: `rm`



Syntax: **rtc** [-1] [-s] *clock_type*

Options: **-1** (the lowercase letter “e”) Set hardware time to local time, not Coordinated Universal Time (UTC). UTC is the standard term for Greenwich Mean Time (GMT).

-s Set hardware to current date and time.

clock_type One of the following:
 hw (hardware clock)
 net [*node*] (a node on a QNX network)

Examples: **rtc hw** Get date from hardware clock.
 rtc -s hw Set hardware clock with date.
 use rtc Display supported clock types.
 rtc net Get date from the current date on node 1.
 rtc net 5 Get date from the current date on node 5.
 rtc -s net 5 Set date on node 5 to the current date on the current node.

Description: The **rtc** command gets or sets the date and time from a battery backed-up hardware clock or from another QNX node.

If your machine has a built-in clock/calendar you should include the following command in your **/etc/config/sysinit** file:

```
rtc hw
```

If the time in your hardware clock is incorrect (perhaps the battery has been replaced), you can set the time using the **-s** (set) option. First, set the QNX time with the **date** utility. Then, issue the appropriate **rtc** command. For example:

```
rtc -s hw
```

You can use clock type **net** [*node*] to get the date from a specified *node*, or to set the date on a specified *node*. If *node* isn’t specified, the default is node 1. When clock type **net** [*node*] is used, the **-1** option has no effect.

rtc

set or get date from realtime clock (QNX)

Exit status: 0 Successful.
 >0 An error occurred.

See also: `clock`, `date`

Syntax: **sac** [**-i** *factor*] [**-n** *node*] [**-p** *priority*] [**-s** *seconds*]
 [<*file*] [>*file*]

Options:

- i** *factor* Use this integration factor (0 to 12; disable = 0).
- n** *node* Gather statistics on *node* (default is the local node).
- p** *priority* Run at this priority (default is 10).
- s** *seconds* Use this interval between snapshots (default is 1.0).
- <*file* Run previously saved log file.
- >*file* Save data to a log file.

Examples: Take a snapshot of an active system with an integration factor of 8:

```
sac -i 8
```

Start a realtime monitor of the dynamics of the system:

```
sac -i 3 -s .1
```

Monitor node 8 using its current integration factor:

```
sac -n 8
```

Description: The **sac** utility is a system activity monitor. It displays a bar graph showing the amount of processor activity at each priority level. If you run **sac** on an idle system, you would see one large bar at priority 0 (priority 1 if **clock** is running).

The kernel collects statistics on processor activity at each priority level. These statistics are integrated over time. The **-i** option sets the integration factor for the system. Smaller values average over shorter periods, resulting in a faster response to changes in system activity. Larger values let you monitor average activity over a longer period of time. If you set the integration factor to 0, then statistics won't be gathered. If you don't specify the **-i** option, then the current integration factor will be used. If a node is not gathering statistics, you will be told to set a value for the integration factor. In this case, re-run **sac** with a nonzero value for the **-i** option.

sac

display system activity at each priority level (QNX)



Note that the **-i** option to **sac** doesn't affect process scheduling. It affects only the statistics gathered for display.

By default, **sac** takes a snapshot of the system statistics once every second. You can adjust this using the **-s** option, which takes an argument in seconds and fractions of seconds. For example, **-s 0.5** would snap every half second. To stop **sac** after the next snap, press <Enter>.

If you specify a file for the standard output, **sac** will write a log of system activity to the file. You may play the log back by invoking **sac** with its standard input redirected.

See also: **sin**

Caveats: On slow machines, **sac** may itself affect priority loading when it runs.

Syntax: `sed [-n] script [file...]`

`sed [-n] [-e script]... [-f script_file]... [file...]`

Options:

- `-e script` Uses the command line *script* for editing *files*. If you specify multiple `-e` options, the *scripts* are applied in the order specified to each line of the input files. If an `-f` option is specified in addition to `-e`, lines are acted upon by *scripts* first.
- `-f script_file` Use the file *script_file* as the script of editing instructions. If multiple `-f` options are specified, the *scripts* are applied in the order specified to each line of the input files. If an `-e` option is specified in addition to `-f`, lines are acted upon by *scripts* first.
- `-n` Suppress the default output, which passes each line to standard output after it's examined for editing. Only lines explicitly selected for output are written.
- file* The pathname of a text file whose contents are read and edited. If you specify multiple files, the files are read in the order specified and the concatenation is edited. If no files are specified, the standard input is used.
- script* A *script* consists of one or more editing instructions that you enter on the command line.

Examples: In the file **myfile**, find and output only those lines containing the string **"tom "**:

```
sed -n -e "/tom /p" myfile
```

In the file **myfile**, replace all occurrences of the string **beneath** with the string **below**, and output to the file **newfile**:

```
sed -e "s/beneath/below/" myfile >newfile
```

Description: The **sed** utility is a stream editor that reads one or more text files, makes editing changes according to editing commands that you specify, and writes the results to standard output.

The **sed** commands are usually stored in a program file (*script_file*), although you may give simple **sed** commands from the command line. By default, **sed** copies files from the file list to its standard output, editing the lines in the process. Conceptually, there is *one* input file, which is the concatenation of all input files specified.

Lines are selected for editing based on their position within the input file, or by pattern matching. If no files are listed, input is taken from standard input (this is the only time standard input is used). The **sed** utility initially reads all the editing commands from all specified sources and places them in an internal table in the order specified. The utility then processes the (concatenated) input file as follows:

- 1) Read one line from the input file and copy to the *pattern space* (a work area).
- 2) For each command that selects the line, act on the pattern space as the edit command specifies, cyclically placing the result into the pattern space.
- 3) After all commands have been checked against the pattern space, write the pattern space to the standard output, provided **-n** was *not* specified. The pattern space may contain zero, one, or several lines at this time.
- 4) Delete the contents of the pattern space.
- 5) Repeat from step 1 until all of the (concatenated) input file has been read and processed.

Scripts A *script* consists of editing commands, one per line, of the following form:

[*address* [,*address*]]*function* [*arguments*]

The *script* of editing commands can be given in the command line, or can be contained in the file *script_file*.

In default operation, **sed** cyclically copies a line of input into a pattern space (unless there is something left after a **D** command), applies in sequence all commands whose addresses select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **-n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval. The *pattern* and *hold spaces* are each limited to 20K.

Addresses

An address is one of the following:

- a decimal number that counts input lines cumulatively across files
- a \$ token that addresses the last line of input
- a context address
- a regular expression

A command line with no address selects every pattern space. A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second address. (If the line number of the second address is than or equal to the line number first selected, then only the first line is selected.) Starting at the first line following the selected range, **sed** looks again for the first address. Thereafter the process is repeated.

Editing commands can be applied to non-selected pattern spaces by use of the negation character (!). For more information, see the section on “Editing commands.”

Regular expressions

The **sed** utility uses basic regular expressions (*REs*), with the following additions:

- in a context address, the construction `\?RE?` (where ? is any character) is mapped to `/RE/`. Note that in the context address `\xabc\xdefx`, the second **x** stands for itself, so that the regular expression is **abcxdef**
- the escape sequence `\n` matches a **<newline>** embedded in the pattern space
- a period (.) matches any character except the last **<newline>** of the pattern space

Editing commands In the following list of functions, the maximum number of permissible addresses for each function is indicated by one of the following:

- [0addr] represents zero addresses
- [1addr] represents zero or one address
- [2addr] represents zero, one, or two addresses

The argument *text* consists of one or more lines. Each embedded **<newline>** in the text must be preceded by a backslash. Other backslashes in text are treated like the backslashes in the replacement string of an **s** editing command, and can be used to protect initial **<blank>**s against the stripping that is done on every script line.

The **r** and **w** editing commands take an optional *rfile* (or *wfile*) parameter, separated from the command letter by zero or more **<blank>**s.

The arguments *rfile* or *wfile* terminate the command line. Each *wfile* is created before processing begins. There can be at most ten distinct *wfile* arguments in the script.

The **b**, **r**, **s**, **t**, **w**, **y**, **!**, and **:** editing commands take additional arguments. The following synopses indicate which arguments are separated from the commands by **<blank>**s:

- [2addr] { *command_list* }
Executes *command_list* only when the pattern space is selected.
- [1addr] **a** \
text
Writes *text* to the standard output after the pattern space is written.
- [2addr] **b** *label*
Branches to the **:** (colon) command bearing the *label*. If *label* is empty, branch to the end of the script.
- [2addr] **c** \
text
Deletes the pattern space. With 0 or 1 address or at the end of a 2-address range, places *text* on the output.

<code>[2addr]d</code>	Deletes the pattern space and starts the next cycle.
<code>[2addr]D</code>	Deletes the initial segment of the pattern space through the first <newline> and starts the next cycle.
<code>[2addr]g</code>	Replaces the contents of the pattern space by the contents of the hold space.
<code>[2addr]G</code>	Appends the contents of the hold space to the pattern space.
<code>[2addr]h</code>	Replaces the contents of the hold space by the contents of the pattern space.
<code>[2addr]H</code>	Appends the contents of the pattern space to the hold space.
<code>[1addr]i\</code> <i>text</i>	Writes <i>text</i> to the standard output before the pattern space is written.
<code>[2addr]l</code>	Lists the pattern space on the standard output in an unambiguous form. Nonprinting characters are listed as hexadecimal digit pairs, with a preceding backslash, with the following exceptions:

<alert>	\a
<backslash>	\\
<backspace>	\b
<carriage return>	\r
<form-feed>	\f
<newline>	\n
<tab>	\t
<vertical tab>	\v

Long lines are folded; the length at which folding occurs is unspecified, but should be appropriate for the output device.

sed

stream editor (POSIX)

<code>[2addr]n</code>	Copies the pattern space to the standard output and replaces the pattern space with the next line of input.						
<code>[2addr]N</code>	Appends the next line of input to the pattern space, using an embedded <code><newline></code> to separate the appended material from the original material. Note that the current line number changes.						
<code>[2addr]p</code>	Copies (prints) the pattern space to the standard output.						
<code>[2addr]P</code>	Copies (prints) the pattern space, up to the first <code><newline></code> , to the standard output.						
<code>[1addr]q</code>	Branches to the end of the script and quits without starting a new cycle.						
<code>[1addr]r rfile</code>	Reads the contents of the <i>rfile</i> file. The contents are placed on the output before reading the next input line.						
<code>[2addr]s/regular expression/replacement string/flags</code>	Substitutes the <i>replacement string</i> for instances of <i>regular expression</i> in the pattern space. Any character can be used instead of <code>/</code>. The value of <i>flags</i> is zero or more of: <table><tr><td><code>n</code></td><td><code>n</code>=1 to 512. Substitutes for the <i>n</i>th occurrence only of the <i>regular expression</i> found within the pattern space.</td></tr><tr><td><code>g</code></td><td>Globally substitutes for all non-overlapping instances of the <i>regular expression</i> rather than just the first one. If both <code>g</code> and <code>n</code> are specified, <code>g</code> takes precedence.</td></tr><tr><td><code>p</code></td><td>Prints the pattern space if a replacement was made.</td></tr></table>	<code>n</code>	<code>n</code> =1 to 512. Substitutes for the <i>n</i> th occurrence only of the <i>regular expression</i> found within the pattern space.	<code>g</code>	Globally substitutes for all non-overlapping instances of the <i>regular expression</i> rather than just the first one. If both <code>g</code> and <code>n</code> are specified, <code>g</code> takes precedence.	<code>p</code>	Prints the pattern space if a replacement was made.
<code>n</code>	<code>n</code> =1 to 512. Substitutes for the <i>n</i> th occurrence only of the <i>regular expression</i> found within the pattern space.						
<code>g</code>	Globally substitutes for all non-overlapping instances of the <i>regular expression</i> rather than just the first one. If both <code>g</code> and <code>n</code> are specified, <code>g</code> takes precedence.						
<code>p</code>	Prints the pattern space if a replacement was made.						

w <i>wfile</i>	Appends (writes) the pattern space to <i>wfile</i> if a replacement was made.
[<i>2addr</i>] t <i>label</i>	Tests. Branches to the : (colon) command bearing the <i>label</i> if any substitutions have been made since the most recent reading of an input line or execution of a t . If <i>label</i> is empty, branches to the end of the script.
[<i>2addr</i>] w <i>wfile</i>	Appends (writes) the pattern space to <i>wfile</i> .
[<i>2addr</i>] x	Exchanges the contents of the pattern and hold spaces.
[<i>2addr</i>] y / <i>string1</i> / <i>string2</i> /	Replaces all occurrences of collating elements in <i>string1</i> with the corresponding collating element in <i>string2</i> . The lengths of <i>string1</i> and <i>string2</i> should be equal.
[<i>2addr</i>] ! <i>function</i>	Applies the <i>function</i> (or group, if <i>function</i> is {) only to the lines that aren't selected by the addresses.
[<i>0addr</i>] : <i>label</i>	This command does nothing; it bears a <i>label</i> for the b and t commands to branch to.
[<i>1addr</i>] =	Places the current line number on the standard output as a line with its own line number.
[<i>0addr</i>]	An empty command is ignored.
[<i>0addr</i>] #	If a # appears as the first character on any line of a script file, that entire line is ignored (treated as a comment), with the single exception that if the first line of the script file begins with #n , the default output is suppressed (as when the -n option is specified on the command line).

Exit status: 0 Successful completion.
 >0 An error occurred.

sed

stream editor (POSIX)

Consequences of errors:

If one or more of the input files (this doesn't include script files) can't be opened for reading, **sed** continues to process the remaining files.

Files:

All files are text files. The *script_files* named by the **-f** option consist of editing commands, one per line. Any number of additional text input files may be specified by the **r** command for insertion of *unedited* data to the standard output at points predetermined by editing rules. A maximum of 10 additional output files may be specified through the use of the **w** command in the script.

See also:

awk, grep, vedit

Syntax: `sh [-c string] [-i] [shell_script] [arg...]`

Options: `-c string` Read commands from *string*.
`-i` Be interactive.

Examples: Invoke a subshell from the command line:

```
sh
```

Invoke a subshell with a command file:

```
sh /etc/backup
```

Invoke a subshell with a command string:

```
sh -c "cd /etc; ls -alF"
```

Description: The **sh** utility is a command interpreter that acts as the interface between the user and QNX. This utility provides powerful facilities for interacting with the operating system, including:

- command input/output
- I/O redirection
- remote command execution
- command substitution
- parameter expansion
- filename generation
- a shell programming language

If **sh** is invoked with the `-c` option, the *string* argument is processed as a shell command list.

If the `-i` option is specified, the shell will be invoked in interactive mode. If the shell's standard input and standard output are attached to a console or a terminal, the shell will be invoked in interactive mode even if `-i` isn't specified.

sh

command interpreter (POSIX)

If no options have been specified but there are command-line arguments, the shell is invoked as a general command executor. The first argument is the filename of a shell script; any additional arguments are passed to the shell as positional parameters.

For a complete description of the Shell, refer to the *User's Guide*.

Exit status:	0	Successful completion.
	>0	An error occurred.
Environment variables:	CDPATH	The directory search path used by the cd command.
	HOME	The pathname of the user's home directory.
	IFS	Input Field Separators: a string of characters used for field splitting.
	PATH	The search path for commands.
	PS1	The primary prompt string (defaults to \$)
	PS2	The secondary prompt string (defaults to >)
Files:	/etc/profile	
	\$HOME/.profile	
See also:	<i>User's Guide</i>	

Syntax: **shutdown [-f] please**

Options: **-f** Shutdown fast. Send a SIGPWR signal, but wait only one second before rebooting.

Description: The **shutdown** utility performs a system shutdown, then reboots the system.

The utility sends a SIGPWR signal to all processes running in the system, and then sleeps for 10 seconds. During this time, processes that have elected to catch the SIGPWR signal may perform an organized shutdown of their own. When the sleep period expires, the system reboots.

See also: *Administrator's Guide*



```
Syntax:  sin [options]

sin [options] args|dir|family|fds|flags|memory|
         registers|rtimers|signals|times|users|format
         format_str

sin [options] freemem|gdt|idt|irqs|names|proxies|
         sessions|tree|vcs

sin [options] info|net|versions
```

```
Options: -S {U|a|b|d|g|i|n|p|s|u}
          Sort on the format_str (see below); select a single letter from
          the set.

-g groupname  Get information on this groupname only.
-h            Don't display headers.
-l            Keep requests local.
-m            Get information on me.
-n node       Get information from this node.
-p pid        Get information on this process ID only.
-P program    Get information on this program only.
-s sid       Get information on this session ID only.
-u username   Get information on this username only.
format_str   Any combination of the following characters:

F flags                a arguments
L login name           b blocked
M magic                d directory
N session node ID      f family
P mem pass             g group
S signals              i process ID
T file/start time      l links
U real user/group ID   m size
                       n name
                       p priority
                       r registers
                       s session ID
                       t CPU time
                       u effective user/group ID
```

sin

display system information (QNX)

```
Examples:  sin
           sin -s 0 time
           sin -m
           sin -n 4 info
           sin -n 5 -p 4 registers
           sin format iuUn
           sin -S p
```

Description: The **sin** utility lets you obtain a “snapshot” of the state of the system (system *information*). If present, the keyword after the command lets you select different types of information and different views of how the information is displayed. You need only type the first two characters of a keyword.

The information can be grouped into three classes as follows:

Class	Description
1	Information keyed off a process
2	Information keyed off other resources
3	General information

The first class gives reports on each process in the system. Since this information exceeds what can be displayed on a line, the **format** keyword allows you to select the components you wish displayed. A number of pre-specified reports have been provided to save typing in the various format options. For example:

```
sin          is equivalent to  sin format sinbpm
sin registers is equivalent to  sin format ir
sin family   is equivalent to  sin format snpbif
```

Output from the first class of commands is sorted by process ID. This can be changed by specifying the **-S** option with an alternate *format_str* (e.g. session ID, or perhaps priority).

The second class gives reports on most of the major operating system objects, for example, interrupt handlers and proxies.

The third class gives reports on important but more general information, for example, the versions of the major System Manager processes running.

The **-n** *node* option obtains information on another node in the network. The default is to obtain information on the current node. This option can be applied to all forms of the command except **gdt** and **idt**.

The **-gmp****Psu** options are used only with the first class of commands. Each option acts as a filter reducing the information displayed.

The **-l** (the letter “el”) option is used only with the **vcs** keyword. It prevents **sin** from going off the current node to obtain the names of the processes on the remote end of a virtual circuit.

See also: **ps**



Syntax: **slay** [*options*]... *process_name*...

- Options:**
- P** *prio* Set the processes matching *process_name* to priority *prio*.
 - S** Suppress killing processes that have child processes. A typical use of this command would be within a shell command that shuts down shells on other devices. Setting this option would prevent **slay** from killing those shells that had other processes (such as editors) running. Running **slay** with the **-q** option also set, will cause **slay** to prompt for a forced kill even if the named process has child processes.
 - h** Set a SIGSTOP signal on a process, effectively holding its execution.
 - n** *nid* Search for the specified processes on the remote node *nid*.
 - p** Return the process ID of the first process that bears the specified *process_name*. This option makes **slay** useful for shell scripts and C programs that want to determine the ID of a process without having to resort to elaborate programming. Within a shell program, the usage would be:

```
slay -p process
```

The exit status is the process ID of the specified process. If the specified process doesn't exist, a process ID of zero is returned and a message is output to the screen.

Within a C program, the *system()* library routine could be used to invoke **slay** to obtain the process ID as follows:

```
pid = system("slay -p process_name");
```

- q** Query before dealing with the process, even if only one process was found with a matching name. This option

slay

kill a process by name (QNX)

is useful for viewing the other process information that **slay** presents.

- s** *sig* Define a signal to send. This option causes the signal *sig* to be raised for the processes matching the *process_names*.
- *signal_number*
A signal number specifying which signal to raise on the processes matching *process_names*.
- t** *ttyname* Match only those processes whose name is *process_name*, with *ttyname* as the controlling terminal. If *ttyname* doesn't begin with a slash (/), it is assumed that it starts with the **/dev/** prefix.
- u** Set a SIGCONT signal on a process. If execution of the process was being held by a SIGSTOP signal, execution will begin where it left off. If the process had not previously had a SIGSTOP set upon it, the SIGCONT signal has no effect.
- v** Be verbose; display messages about processes being signaled.

process_name

The name of a process to operate on.

Examples: Kill **spooler** process on node 2:

```
slay -n 2 spooler
```

Kill **dragon** process on **/dev/tty3**, querying first:

```
slay -qt /dev/tty3 dragon
```

Change priority of **test** process to 20:

```
slay -P 20 test
```

Description: You use the **slay** utility to kill a process by name rather than by process ID. This saves you from having to first run the **ps** utility before issuing a **kill** command. Process names are specified without the path. For example, let's say you have a process called **/bin/sleep** that you wish to kill. Entering **sleep** as the process name would be sufficient to allow **slay** to find and kill it.

There are many forms of this command. The simplest and most often used form is:

slay *process_name*

This command locates the process bearing the specified name. If only one is found, a SIGTERM signal is set on it. If more than one process bears the specified name, you are prompted for a yes/no response for each process. When each process is listed in this form, the process name, *pid*, and tty group/member numbers are also displayed to help you make a selection.

See also: **kill**, **ps**



Syntax: **sleep** *time*

time A non-negative decimal integer specifying the number of seconds for which to suspend execution.

Description: The **sleep** utility suspends execution for at least the integral number of seconds specified by the *time* operand.

Exit status: 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal was received.

 >0 An error occurred.

Asynchronous events: The **sleep** utility takes the default action for all signals, except that **sleep** terminates normally with a zero exit status on receipt of a SIGALRM signal.

See also: **sh**



1

sort

sort, merge, or sequence-check text files (POSIX)

Syntax: `sort [-m] [-o name] [-bdfinru] [-t char] [-k keydef]
[+oldkey] [file...]`

```
sort [-c] [-bdfinru] [-t char] [-k keydef] [+oldkey]  
      [file]
```

Options:		
-c	Check that the single input file is already sorted. Produce no output.	
-m	Merge only. It's assumed that the input files are already sorted.	
-o <i>name</i>	The <i>name</i> argument is the name of an output file to be used instead of the default which is standard output. This file can be the same as one of the input files.	
-u	Unique; suppress all but one key in each set of lines having equal keys.	

The following options override the default ordering rules. When ordering options appear independent of key field specifications, the requested field ordering rules are applied globally to all sort keys. When attached to a specific key (see option **-k**), the specified ordering options override all global ordering options for that key.

- b** Ignore leading **<blank>** characters in field comparisons.
- d** Sort in “dictionary order.” Only **<blank>** characters and alphanumeric characters are significant in comparisons.
- f** Fold uppercase letters into lowercase.
- i** Ignore all non-printable characters.
- k** *keydef* Define *keydef* to be a sort key.
- n** Interpret the field as numeric, including the sign and “thousands” separator. Sort in numeric order. This implies the **-b** option.
- r** Reverse the sort order.

sort

sort, merge, or sequence-check text files (POSIX)

-t *char* Use *char* as the field separator character.

file The pathname of a file to be sorted, merged, or checked. If you don't specify any files to be sorted, or if a *file* operand is the dash character (-), the standard input is used.

Examples: The standard password file has the form:

```
username : password : uid : gid : misc : home : shell
```

To sort the password database by *username*:

```
sort -t: /etc/passwd
```

Description: The **sort** utility orders lines from a set of files (or standard input if no files are provided) and merges the output into the specified output (or standard output if no output file is specified). The **sort** utility provides a number of options for controlling the sorting mechanism. The default behavior of **sort** is to treat all lines as a sequence of characters to be sorted in ascending order.

The **sort** utility defines the following structure on files:

- a file is a sequence of lines
- a line is a sequence of fields followed by a **<newline>**
- a field is a sequence of characters followed by a field separator
- a field separator, by default, is a **<blank>**, although this may be altered by the **-t** option

There **-t** option has subtle effects. If the option is *not* used, only the first **<blank>** in a string of **<blank>** characters is interpreted as a field separator; the remaining are considered part of the next field. If the **-t** option is used, every occurrence of the field separator character is considered a terminator for a field. Thus, four adjacent field separators represent three empty fields.

A sort key field may be defined using the following syntax:

```
-k field_start [type_string] [,field_end] [type_string]
```

where *field_start* and *field_end* specify the beginning and end of the key field, and *type_string* is used to specify attributes specific to the key.

The *field_start* and *field_end* are each specified by a pair of digits of the form *m.n*, where the *m* refers to the field starting after the *m*th field separator in a line. For *field_start*, the *.n* refers to the *n*th character of the specified field, and is taken as zero if not specified. For *field_end*, the *.n* refers to the *n*th character after the last character of the specified field, and is taken as zero if not specified.

The *type_string* may be formed from the characters **bdfinr** which apply their defined attributes to the determination of the key.

There is another mechanism for defining sort keys, which remains for historical reasons. The syntax of this mechanism is:

```
+field_start [type_string] -field_end [type_string]
```

The semantics are the same as for the **-k** option.

Let's return to our password database example. The following command would sort the password database by *gid*, then *uid*, then *username*:

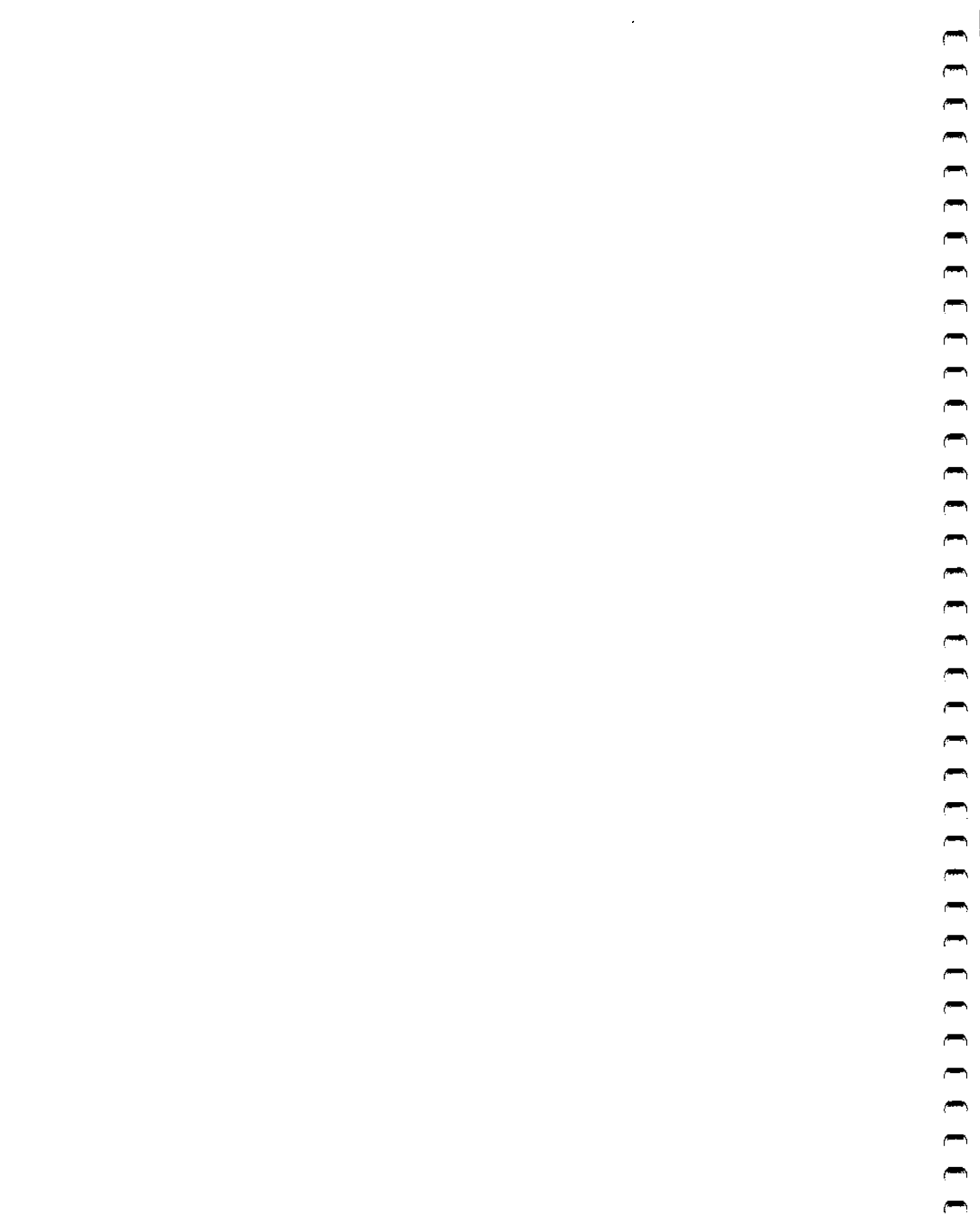
```
sort -t: -k 3.0n -k 2.0n -k 0.0d /etc/passwd
```

Notice that the *type_string* attached to *field_string* is used to determine the type of comparison with this key.

The second *type_string* is solely to adjust the extent of the *sort_key*, thus only **i** and **b** make any sense. Any other characters are ignored.

- Exit status:**
- 0 All input files were output successfully, or option **-c** was specified and the input file was correctly sorted.
 - 1 The **-c** option was used, and the input file was not sorted.
 - >1 An error occurred.

See also: **diff**, **grep**, **sed**



Syntax: `spatch [-bp] filename`

Options:

- b** Browse. Don't allow the **Save** option. The file or disk will be accessed in read-only mode.
- p** Pause before starting.
- filename* The file to be examined.

Examples:

```
spatch /bin/ls
spatch /dev/hd0
```

Description: The **spatch** utility provides full-screen editing of files or disk blocks. The screen displays a 16 by 16 (256) byte image of the data being examined, similar to that shown below:

```

Edit Next Prev Home Goto Find Continue Save Addr Quit
000000000: 2E 28 6E 65 77 29 20 53 50 41 54 43 48 20 22 46  (new) SPATCH "F
000000010: 75 6C 6C 20 73 63 72 65 65 6E 20 70 61 74 63 68  ull screen patch
000000020: 20 75 74 69 6C 69 74 79 22 1E 2E 28 73 79 6E 74  utility"..(synt
000000030: 61 78 29 1E 09 11 73 70 61 74 63 68 10 20 20 11  ax)...spatch. .
000000040: 66 69 6C 65 10 20 20 AE 66 69 6C 65 6E 61 6D 65  file. .filename
000000050: AF 1E 09 11 73 70 61 74 63 68 10 20 20 11 64 69  ...spatch. .di
000000060: 73 6B 10 20 20 AE 64 72 69 76 65 AF 20 20 AE 62  sk. .drive. .b
000000070: 6C 6F 63 6B AF 1E 09 11 73 70 61 74 63 68 10 20  lock....spatch.
000000080: 20 11 6D 65 6D 10 20 20 AE 73 65 67 6D 65 6E 74  .mem. .segment
000000090: AF 20 20 AE 6F 66 66 73 65 74 AF 1E 2E 28 65 78  . .offset...(ex
0000000a0: 61 6D 70 6C 65 73 29 1E 09 11 73 70 61 74 63 68  amples)...spatch
0000000b0: 20 20 66 69 6C 65 20 20 2F 63 6D 64 73 2F 6C 73  file /bin/ls
0000000c0: 1E 09 73 70 61 74 63 68 20 20 64 69 73 6B 20 20  ..spatch disk
0000000d0: 31 20 20 31 1E 09 73 70 61 74 63 68 20 20 6D 65 1 1..spatch me
0000000e0: 6D 20 20 62 30 30 30 20 20 30 10 1E 2E 28 73 74  m b000 0...(st
0000000f0: 61 72 74 29 1E 53 50 41 54 43 48 20 69 73 20 61  art).SPATCH is a

```

The menu at the top of the screen can be used to select one of the options listed below. You can either position the inverse cursor to the desired menu item with the arrow keys and type <Enter>, or simply type the first letter of the command.

- Edit**  Enter the data area. The <Tab> key switches between hex and ASCII data entry. The <Esc> key returns you to the menu. The changed data isn't updated to the disk or memory.
- Next** Move forward 256 bytes. You can also use <PgDn>.
- Prev** Move backward 256 bytes. You can also use <PgUp>.

spatch

full-screen patch utility (QNX)

	Home	Go to the start of the file, disk, or memory. You can also use <Home>.
	Last	Go to the last block of the file or disk. You can also use <End>.
	Goto	You'll be prompted for an address. The type of address will depend on the source of the data (file or disk) and the address type.
	Find	You'll be prompted for a pattern to search for. The pattern may consist of single characters or hex digit pairs separated by a space. For example: 61 62 63 d e a b c d e match the 5 characters "abcde" match the 5 characters "abcde" To stop the search, press any key.
	Continue	Find the next occurrence of the last pattern found. Usually used after a find when searching.
	Save	Save the current screen back to the source. Without issuing this command all changes made using Edit will be lost as soon as you leave the current screen of data. The save option is disabled by the -b (browse) option.
	Addr	Toggle through two address types: Absolute Disk Block:Offset Default for FILE Default for DISK
	Quit	Leave the spatch utility.
Exit status:	0	Successful.
	>0	An error occurred.
Environment variables:	TERM	Interpreted as the name of the terminal type.
See also:	cat, hd	

Caveats: You may not use the **spatch** utility on a disk when there are open files on the disk, unless the **-b** (browse) option is used.

Syntax: `stty [-a|-g] [operands] [< device]`

Options:

- `-a` Display *all* settings.
- `-g` Display in “getable” form.

Examples: Display settings of the terminal to which **stty** is attached:

```
stty
```

Display settings of a specified device:

```
stty < /dev/ser1
```

Change the baud rate of a specified device:

```
stty baud=1200 < /dev/ser1
```

Put terminal into a fixed, sane state:

```
stty +sane
```

Set editing keys to VT100 values:

```
stty term=vt100
```

Get current settings into a shell variable:

```
saveterm = "$(stty -g)"
```

Restore settings from a shell variable:

```
stty $saveterm
```

Description: The **stty** utility sets and/or reports on terminal I/O characteristics for the device that is its standard input. If no operands are specified, **stty** displays the settings. If operands are given, then **stty** will change the terminal state to reflect those settings.

Terminal settings fall into two major categories: *edit mode* or *raw mode*. In edit mode, input data can be edited by the user, and is made available to programs only when a CR is pressed. Output data will normally be presented in a human-readable format. In *raw* mode, all data flows to and from the terminal with little extra processing. If you're at a shell prompt, your terminal will probably be in the default

stty

set tty attributes (POSIX)

system *edit* mode. A full-screen program such as an editor, on the other hand, typically puts the terminal into *raw* mode.

Normally, **stty** displays only significant settings relative to the system default settings for *edit* or *raw*, and will display only the defined control characters. If **-a** or **-g** are specified, then **stty** will display *all* the settings.

The **stty** utility manages a very large number of potential terminal attributes and control characters. Most of these parameters are very device-specific and will seldom need to be changed by the user. Programs often change these terminal attributes in the course of their operation, and upon occasion (such as abnormal termination) may leave the terminal settings in an unknown state. The **stty +edit** option is a convenient method of restoring a terminal to a usable state.

Supported operands:

The complete list of operands supported by **stty** are provided in the following tables. In these tables, the following conventions are used:

- number* represents a decimal integer number (e.g. 9600).
- name* represents a string of characters (e.g. vt100).
- value* can be a single character (e.g. ~) or a 2-digit hexadecimal number (e.g. 1B) or one of the following character pairs:

^~	00 (undefined)
^A to ^Z	01 to 1A
^[	1B
^\	1C
^]	1D
^^	1E
^_	1F
^?	7F

An option that starts with $\pm$ can start with either of these characters:

- + (turn the option on)
- (turn the option off)

If neither + nor - is specified, + is assumed.

Also note that the = character is optional in operands of the form *<keyword>=<value>*.



The following descriptions give the action taken when the option is on.

Line control parameters

baud=number	define input and output baud rates
ispeed=number	define input baud rate
ospeed=number	define output baud rate
par=none	no parity (same as -parenb)
par=odd	odd parity (same as +parenb , +parodd , -parstk)
par=even	even parity (same as +parenb , -parodd , -parstk)
par=mark	mark parity (same as +parenb , +parodd , +parstk)
par=space	space parity (same as +parenb , -parodd , +parstk)
bits=5	5-bit characters
bits=6	6-bit characters
bits=7	7-bit characters
bits=8	8-bit characters
stopb=2	2-stop bits
stopb=1	1-stop bits
±parenb	enable parity
+parodd	odd parity
-parodd	even parity
±parstk	stick parity
+cs5	same as bits=5
+cs6	same as bits=6
+cs7	same as bits=7
+cs8	same as bits=8
+cstopb	same as stopb=2
-cstopb	same as stopb=1
number	same as baud=number
+evenp	same as par=even , bits=7
-evenp	same as par=none , bits=8
+parity	same as par=even , bits=7
-parity	same as par=none , bits=8
+oddp	same as par=odd , bits=7
-oddp	same as par=none , bits=8

stty

set tty attributes (POSIX)

	±hupcl	hangup on last close
	±hup	same as hupcl
	±cread	enable receiver
	±clocal	assume no modem control
	±ihflow	enable hardware input flow control
	±ohflow	enable hardware output flow control
	±isflow	enable software input flow control
	±osflow	enable software output flow control
Input processing parameters	±ignbrk	ignore received hardware breaks
	±brkint	generate SIGINT upon break
	±ignpar	ignore parity errors
	±parmrk	mark parity errors
	±inpck	enable software parity checking
	±istrip	strip 7th bit from received characters
	±inlcr	map NL into CR on input
	±igncr	ignore received CR
	±icrnl	map CR into NL on input
	±ixon	same as osflow
	±ixoff	same as isflow
	±isig	generate signals upon receipt of special characters
	±icanon	enable input line editing
	±iexten	enable “extra” special characters
	±echo	echo received characters
	±echoe	erase character will erase displayed character
	±echok	kill character will erase displayed line
	±echonl	echo NL, even if ECHO is off
	±noflsh	don’t flush I/O after INTR, QUIT, or SUSP
	+flush	flush input and output data immediately
	min=<i>number</i>	minimum characters required to satisfy <i>raw</i> input
	time=<i>number</i>	timeout value for <i>raw</i> input
	+nl	same as +icrnl
	-nl	same as -icrnl , -inlcr , -igncr
	+sane	reset all parameters to sane values based on current mode (<i>edit/raw</i>)
	+fix	same as +sane
	+edit	reset parameters to system default <i>edit</i> mode
	+raw	reset parameters to system default <i>raw</i> mode

Output processing parameters	±opost	post-process output data
Special control characters	eof= <i>value</i> eol= <i>value</i> erase= <i>value</i> kill= <i>value</i> intr= <i>value</i> quit= <i>value</i> susp= <i>value</i> stop= <i>value</i> start= <i>value</i> +ek	end-of-file character end-of-line character delete previous character delete entire line character generate SIGINT character generate SIGQUIT character generate SIGTSTP character stop output resume output resets ERASE and KILL to system defaults
Extended line-editing character sequences	QNX supports multi-character sequences to support editing capabilities in addition to the basic “erase” and “kill” functions. All of these sequences are assumed to start with an (up to) 4-character <i>prefix</i>, a single-character <i>action</i>, followed by an (up to) 4-character <i>suffix</i>. Typically, the cursor keys on your terminal will be used for these functions.	
	+load	set editing keys based on currently defined terminal type
	term= <i>name</i>	set editing keys for the specified type of terminal
	pr1= <i>value</i>	first character of <i>prefix</i>
	pr2= <i>value</i>	second character of <i>prefix</i>
	pr3= <i>value</i>	third character of <i>prefix</i>
	pr4= <i>value</i>	fourth character of <i>prefix</i>
	sf1= <i>value</i>	first character of <i>suffix</i>
	sf2= <i>value</i>	second character of <i>suffix</i>
	sf3= <i>value</i>	third character of <i>suffix</i>
	sf4= <i>value</i>	fourth character of <i>suffix</i>
Action characters	The following are <i>action</i> characters when preceded by <i>prefix</i> and followed by <i>suffix</i>.	
	up= <i>value</i>	recall previous line
	down= <i>value</i>	recall next line
	left= <i>value</i>	move cursor left
	right= <i>value</i>	move cursor right

stty

set tty attributes (POSIX)

	ins=value	toggle insert mode
	del=value	delete current character
	rub=value	delete previous character
	can=value	delete entire line character
	home=value	move cursor to beginning of line
	end=value	move cursor to end of line
Console-specific options	±numlock	keyboard is in Num Lock mode
	±capslock	keyboard is in Caps Lock mode
	±scrlock	keyboard is in Scroll Lock mode
	±noboot	disable keyboard reboot
	±noswitch	disable keyboard console switching
	±nodebug	disable keyboard debugger entry
	±noresize	disable keyboard resizing
	±nohotkey	disable keyboard hotkeys
	±nocolor	disable colours on console display
	±monocurs	force cursor to monochrome adapter values
Serial port-specific options	±scanmode	put keyboard into <i>scancode</i> mode
	±DTR	turn on data-terminal-ready hardware handshaking line
	±RTS	turn on request-to-send hardware handshaking line
	±BRK	force transmit data line to break condition



The *Administrator's Guide* gives more detailed information concerning the circumstances under which many of these options will be required.

The **stty** utility usually displays as much information about a device as it can. Unless the **-g** option is set, the following settings may also be displayed, depending on the type of device:

Serial port information

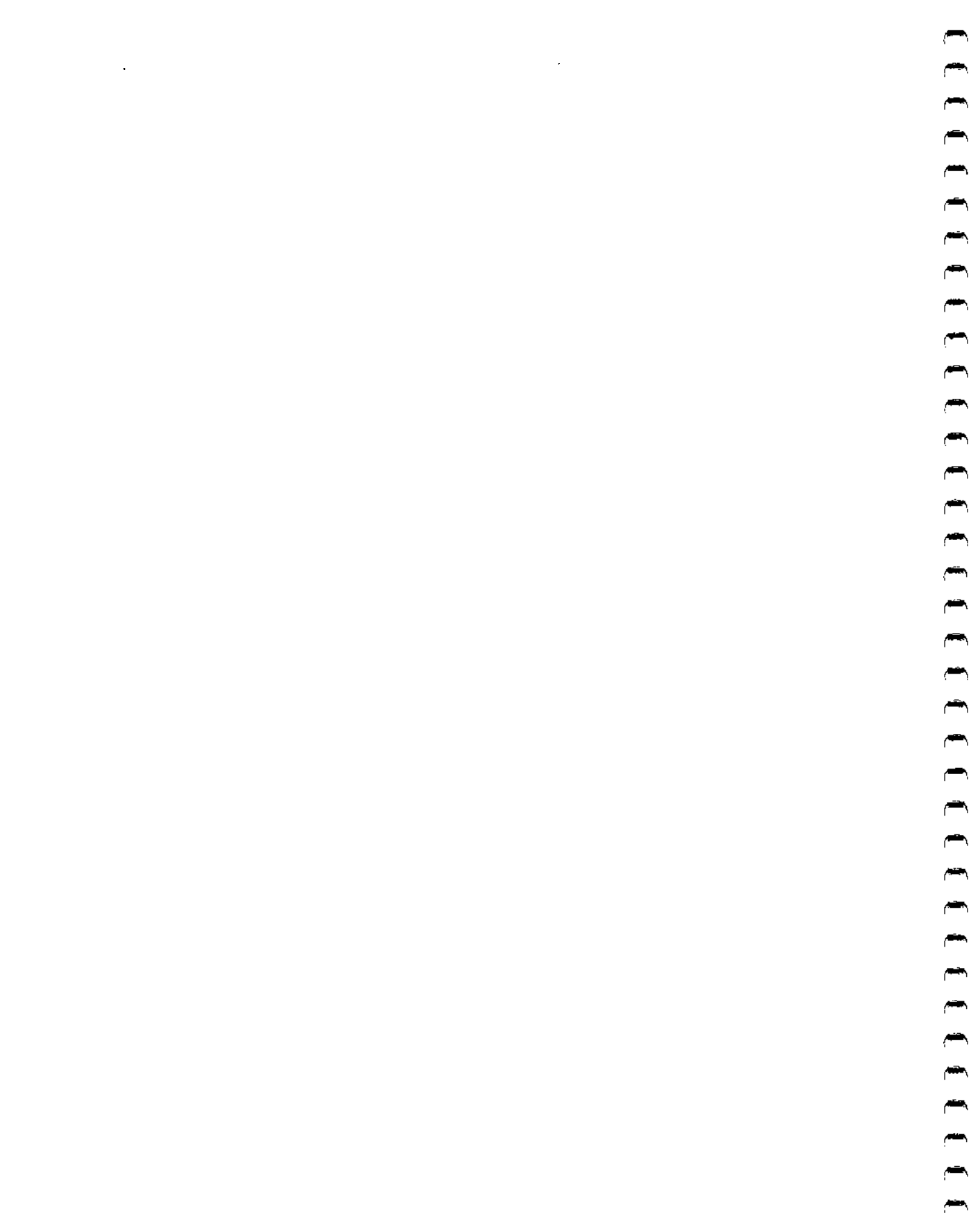
±cts	clear-to-send line is on
±dsr	data-set-ready line is on
±ri	ring indicator is on
±cd	carrier detect is on

Parallel port information

±ERR	error condition is set
±ONL	printer is online
±PE	paper-empty indicator
±BSY	printer is busy

Exit status: 0 The utility executed successfully.
 >0 An error occurred.

See also: *Administrator's Guide*



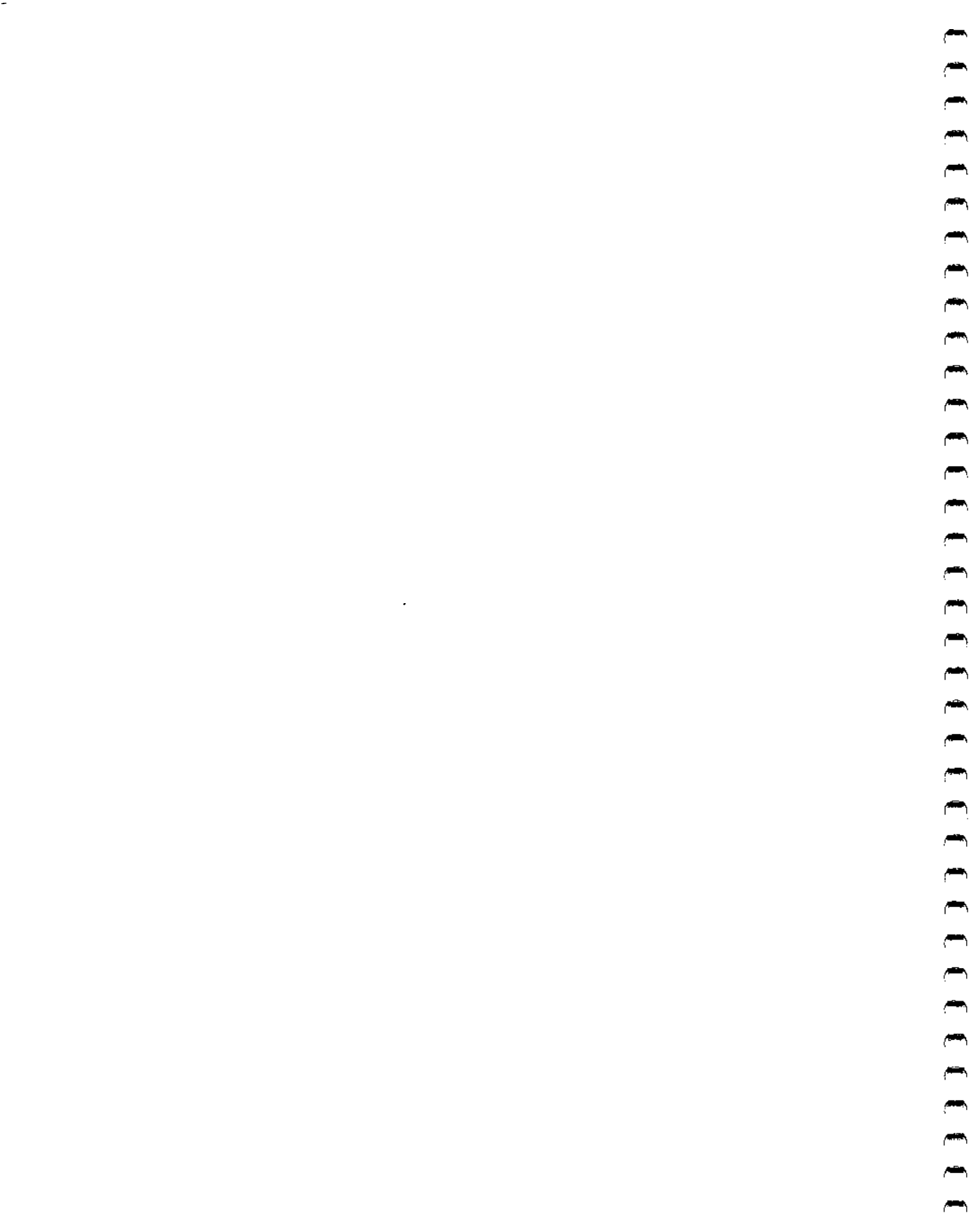
Syntax: **su** [*userid*]

Description: The **su** utility requests the password of the given *userid* (root, by default) and changes to that *userid*, invoking its shell but by modifying only essential elements of the environment. Only the **HOME**, **PATH**, and, possibly, **SHELL** environment variables are changed, but the new shell has the rights and privileges of the user specified. The new *userid* remains in effect until this shell exits.

The **su** utility is intended for users to be able to temporarily become the superuser to perform a specific function, then return to their regular *userid*.

Files: **/etc/passwd**

See also: **login, passwd**



Syntax: **tail** [-c|-l] [-f] [-n *number*] [*file*]

Options:

- c** Measure the quantity of output in bytes.
- f** If the input file is a regular file (i.e. not a tty or FIFO), don't terminate after the last line of the input file has been copied, but enter a continuous loop. The **tail** utility then sleeps for approximately one second, then attempts to read and copy further bytes from the input file.
- l** (the lowercase letter "l") Measure the quantity of output in lines; this is the default unit of measure.
- n *number*** Copy output using this number of lines (default) or bytes. The *number* argument is a decimal integer whose sign specifies the location in the file to begin copying:

 If the sign is: then copying starts relative to the:
 + beginning of the file
 - end of the file
 none end of the file

 If you omit the **-n *number*** option, the default is **-10**, the last ten units (lines or bytes) of the file.
- file*** The pathname of an input file. If you don't specify a file, the standard input is used.

Examples: You can use the **-f** option to monitor the growth of a file that is being written by a process. For example, the command:

```
tail -f fred
```

prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and terminated. As another example, the command:

```
tail -cf -n 15 fred
```


tail

copy the last part of a file (POSIX)

prints the last 15 bytes of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and terminated.

Description: The **tail** utility copies its input file to the standard output, beginning at a designated place.

Copying begins at the point in the file indicated by the **-n number** option. The argument *number* is counted in units of lines or bytes, according to the **-l** and **-c** options.

When **tail** is applied to a non-seekable file (e.g. a tty), **tail** must maintain an internal buffer. This buffer is large enough to hold *at least* 10 lines of characters.

Exit status: 0 Successful completion.
>0 An error occurred.

See also: **cat**, **head**, **more**

Syntax: `textto -c|l|r [-z] file...`

Options:

- `-c` Convert the specified files to DOS format (CR/LF).
- `-l` (the lowercase letter “e”) Convert the specified files to QNX 4.0 format (LF).
- `-r` Convert the specified files to QNX 2.1 format (RS).
- `-z` Strip all `^Z` codes from the file.

file The name of the file to be converted. Wildcards may be used.

Examples: Change any QNX 2.1 record separators in `prog.c` to QNX 4.0 line feeds:

```
textto -l prog.c
```

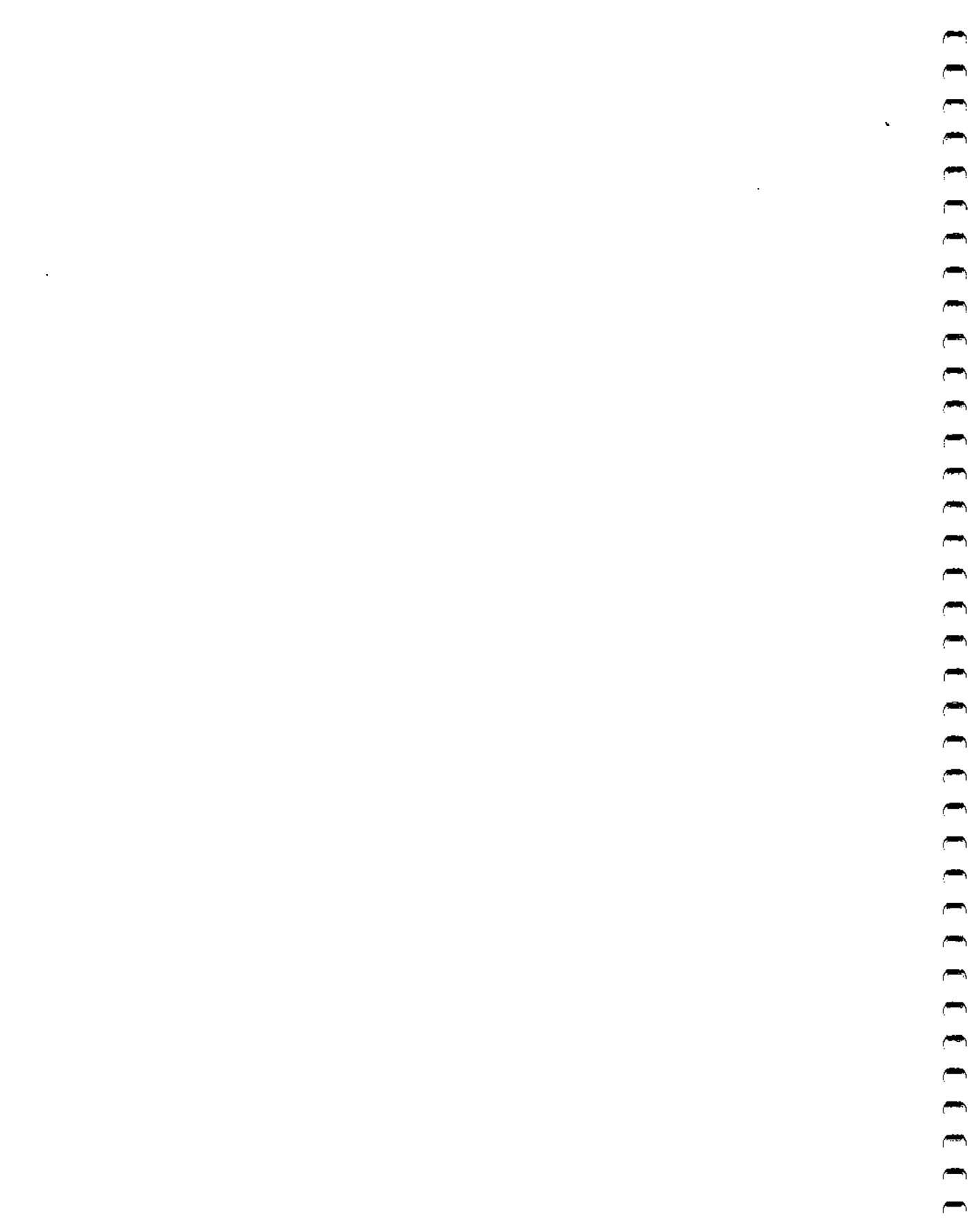
Change any QNX 4.0 line feeds in all of the `.h` files in the current directory to QNX 2.1 record separators:

```
textto -r *.h
```

Change any QNX 4.0 line feeds in all of the `.txt` files in the current directory to DOS CR/LFs:

```
textto -c *.txt
```

Description: The `textto` utility lets you convert text files to QNX 2.1, QNX 4.0, or DOS format. The `-z` option can be used to strip `^Z` codes from DOS-based files. Some DOS programs use this character as an end-of-file indicator.



Syntax: `tic [-c] [-v[n]] file`

Options:

- `-c` Perform only a syntax check of the input file. No output file is produced.
- `-v[n]` Set verbose output level. Progress information is output to **stderr** as **tic** works through the source file. The optional integer argument controls the verbosity of the report, **1** providing the least detail and **10** providing the most. The default is **0**, providing “quiet” compilation.
- file* The pathname of a file that contains one or more terminal descriptions in **terminfo** source format (see the Terminfo Technical Note for source format details). Each description in the file describes a particular terminal. When a **use=entry_name** field is encountered within a terminal description, **tic** reads the previously compiled description from the **/usr/lib/terminfo** directory to finish the entry. This allows terminal descriptions that are only slightly different from one previously defined to be expressed as an extension to that previous description. If the environment variable **TERMINFO** is set, the directory named will be searched instead of the **/usr/lib/terminfo** directory.

Description: The **tic** utility translates a **terminfo** file from the source format into the compiled format, ready to use by applications running on the terminal type named in the **terminfo** source file. The compiled format is used by the QNX *term()* functions, the *terminfo* functions, and the *termcap* functions.

The compiled **terminfo** capability files are stored in single-character directories, under the **/usr/lib/terminfo** directory. For example, the VT100 **terminfo** file is stored in: **/usr/lib/terminfo/v/vt100**.

If the environment variable **TERMINFO** is defined, the compiled output from **tic** will be placed under the directory named instead of in the **/usr/lib/terminfo** directory.

tic

terminfo compiler (UNIX)

Most error messages produced by **tic** will indicate the line of the source file in error and the terminal name being processed at that point.

See also: **infocmp**, *Terminfo Technical Note*

Caveats: The total size of a compiled **terminfo** description cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names longer than 14 characters will be truncated to 14 characters.

Syntax: `ticksize [size]`

Options: `size` Time duration for a clock tick. Valid numbers are .5, 1, 2, 5, 10, 25, or 50 milliseconds.

Examples:

<code>ticksize</code>	Query current tick size
<code>ticksize 50</code>	Set the tick size to 50 milliseconds
<code>ticksize .5</code>	Set the tick size to .5 milliseconds

Description: The `ticksize` utility queries or changes the rate at which timer interrupts (called ticks) are applied to the system. The tick size determines the *granularity* of all software system timers. All time requests will be rounded up to this granularity. For example, if the tick size was 10 milliseconds, a request to wait for 1 millisecond may wait for up to 10 milliseconds.

Setting the tick size will not affect the system date, nor will it affect active software timers, other than the granularity of the point where they are triggered.

QNX uses a realtime preemptive scheduler. Therefore, if a process becomes READY (perhaps as the result of an external interrupt) and is the highest priority process, it will be scheduled immediately. There is no delay based upon the tick size. Ticks are used for software timers only.

If your machine has a CPU speed rating of less than 600 as reported by:

```
sin info
```

then you cannot set your tick size below 1 millisecond.

When QNX boots, it sets the tick size to 10 milliseconds for machines with a CPU speed greater than or equal to 300. For machines with CPU speeds less than 300, the tick size is set to 50 milliseconds.

ticksize

set the timeslice rate (QNX)

Exit status: 0 The tick size was successfully queried or changed.
 >0 An error occurred. The tick size will not be changed.

See also: `sin`

Syntax: **time** *command*

command The command, including any of its options, for which the execution time will be determined.

Examples: **time dcheck -b2048 /dev/hd0**

Description: The **time** utility times the execution time of *command*, which must be specified. When execution of *command* is complete, the **time** utility displays the total elapsed time for execution, the time spent in the system, and the time spent in executing *command*. Times are reported in hours, minutes, and seconds.

Times are not always accurate to the milliseconds as displayed, depending on the tick size.

Exit status: The **time** utility exits with the exit status of the *command* that was timed.

See also: **ticksize**

Caveats: The **time** utility does not allow *command* to be executed on another CPU. For example, the following is invalid:

```
time //61 sleep 30
```

Instead of the above, you can use the following:

```
//61 time sleep 30
```




Syntax: `tinit [-T device...] [-c command] [-f optionfile]
[-m minutes] [-t device...] &`

Options:

- `-T device` Start *command* on this device when any character is input.
- `-c command` Start *command* on the devices specified by the `-t` or `-T` option. The default is **login**. This affects all `-t` and `-T` options to the right on the command line.
- `-f optionfile` The specified text file contains more command-line options. This is useful in systems with large numbers of options (typically `-c` or `-t` options) that would result in long and complex command lines.
- `-m minutes` If the line is idle (no input) for the indicated number of minutes, set a **SIGHUP** signal on the process group of that controlling terminal. If *minutes* is zero (the default), the idle timeouts are disabled. This affects all `-t` and `-T` options to the right on the command line.
- `-t device` Start *command* immediately on this device.

Examples:

```
tinit -T /dev/con* -t con1 &
tinit -t con1 con2 -c "modem -b 19200 -L" -t ser1 &
tinit -f /etc/config/tinit &
```

Description: The **tinit** utility lets you bring up **login** (or other programs) on devices. You normally use **tinit** to invoke **login** on the console(s) and serial terminals. It runs as a background process, and is nearly always started in the **sysinit** file.

After creating the specified *commands* as its children, **tinit** waits for any of them to terminate. When one of these commands terminates, **tinit** re-invokes the command on that device.

For example, let's assume that **tinit** started a **login** on `/dev/con1`. After logging in, the user is presented with a shell.

Then, after executing any number of commands, the user decides to terminate the shell. At this point, **tinit** detects the death and starts a new **login** on **/dev/con1**.

The list of terminal devices on which you want the command to be started can be specified by one or more **-T** or **-t** options. If a device appears more than once, the utility will use the parameters in effect for the last use on the command line. You may use a separate **-T** or **-t** for each device or use spaces in order to specify several devices:

```
-t con2 -t con1
```

OR

```
-t con2 con1
```

If the device name doesn't start with a slash, it's assumed the name starts with the following:

```
/dev/
```

The **tinit** utility scans its options left to right. Each **-c** replaces the currently defined command and each **-m** replaces the currently defined timeout. The **-t** and **-T** options memorize the current settings for starting a command on a device. For example:

```
-t con1 -c modem -m 10 -t ser1
```

would start **login** (default **-c**) with no timeout (default **-m**) on **/dev/con1**, then start **modem** with a 10-minute timeout on **/dev/ser1**.

For devices specified with the **-T** option, **tinit** doesn't start the command right away. Instead, it waits for any character to be received from the device.

If you want to start several different commands on many devices, the command line for **tinit** may become very long. You can use the **-f** option to specify a file that contains options. Unlike command-line options, which are expanded by the shell, the options in this file are taken as is. For example, **-t /dev/con*** in a file will *not* be expanded.

If you want to group several blank-separated words into a single argument, enclose them in double quotes (").

Here's an example of a possible option file:

```
-t con1 -T con2 con3 con4
-c "modem -b 19200 -L" -m 10 -t ser1 ser2
-c "modem -b 1200" ser3
```

You can mix options on the command line and file options. File options are always processed *after* command-line options.

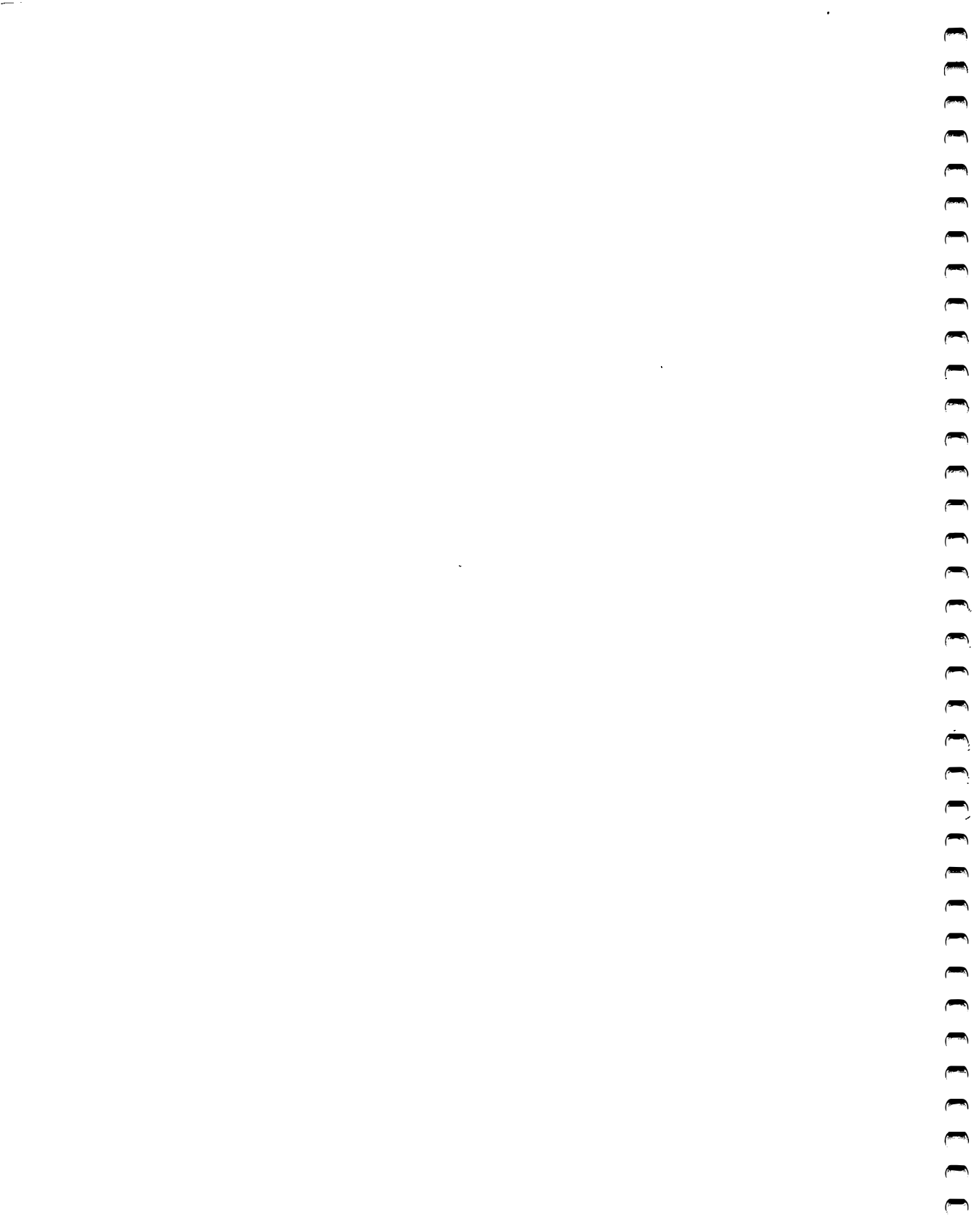
If **tinit** tries to start or restart a command, and the command fails to load, **tinit** retries once every minute until it is successful.

The **tinit** utility will log each command invocation to the file **/etc/acclog**, provided the file exists. For details, see the *Administrator's Guide*.

Files: **/etc/acclog** (accounting records are appended to this file)

See also: **login, modem**

Administrator's Guide: **sinit**



Syntax: `touch [-acm] [-r ref_file|-t time] file...`

- Options:**
- a** Change the access time of *file* to *time*, or to the current time if *time* isn't specified. Don't change the modification time unless the **-m** option is also specified.
 - c** If *file* doesn't already exist, don't create *file* and don't write any diagnostic messages concerning this condition.
 - m** Change the modification time of *file* to *time*, or to the current time if *time* isn't specified. Don't change the access time unless **-a** is also specified.
 - r *ref_file*** Use the corresponding reference file's modification time instead of the current time.
 - t *time*** Use the specified *time* instead of the current time, where *time* is a decimal number of the form:

`[ [CC]YY]MMDDhhmm[.SS]`

The two-digit pairs in *time* represent the following:

<i>CC</i>	the first two digits of the year (i.e. the century)
<i>YY</i>	the second two digits of the year
<i>MM</i>	the month of the year (01-12)
<i>DD</i>	the day of the month (01-31)
<i>hh</i>	the hour of the day (00-23)
<i>mm</i>	the minute of the hour (00-59)
<i>SS</i>	the second of the minute (00-61)

Both *CC* and *YY* are optional. If you specify neither of these, the current year is assumed. If you specify *YY*, but you don't specify *CC*, *CC* is derived as follows:

If *YY* is: *CC* becomes:

69-99	19
00-68	20

The resulting time is affected by the value of the **TZ** environment variable (see below). If the resulting time

touch

change file access and modification times (POSIX)

precedes 0 hours, 0 minutes, 0 seconds, January 1, 1970 Coordinated Universal Time (i.e. “The Epoch”), **touch** exits immediately with an error status.

The range for *SS* is 00-61 rather than 00-59 to allow leap seconds. If *SS* is 60 or 61, and the resulting time—as affected by the **TZ** environment variable—doesn’t refer to a leap second, the resulting time is one second after a time where *SS* is 59. If you don’t give a value to *SS*, it’s assumed to be 0.

file The pathname of a file whose times are to be modified.

Examples: Set **file1**’s access and modification time to the current system time:

```
touch file1
```

Change **file2**’s access and modification time to be the same as **file1**’s modification time:

```
touch -r file1 file2
```

Set **file3**’s access and modification time to December 25, 12:34, using the current year:

```
touch -t 12251234 file3
```

Description: The **touch** utility lets you change either the modification times or access times of files, or both.

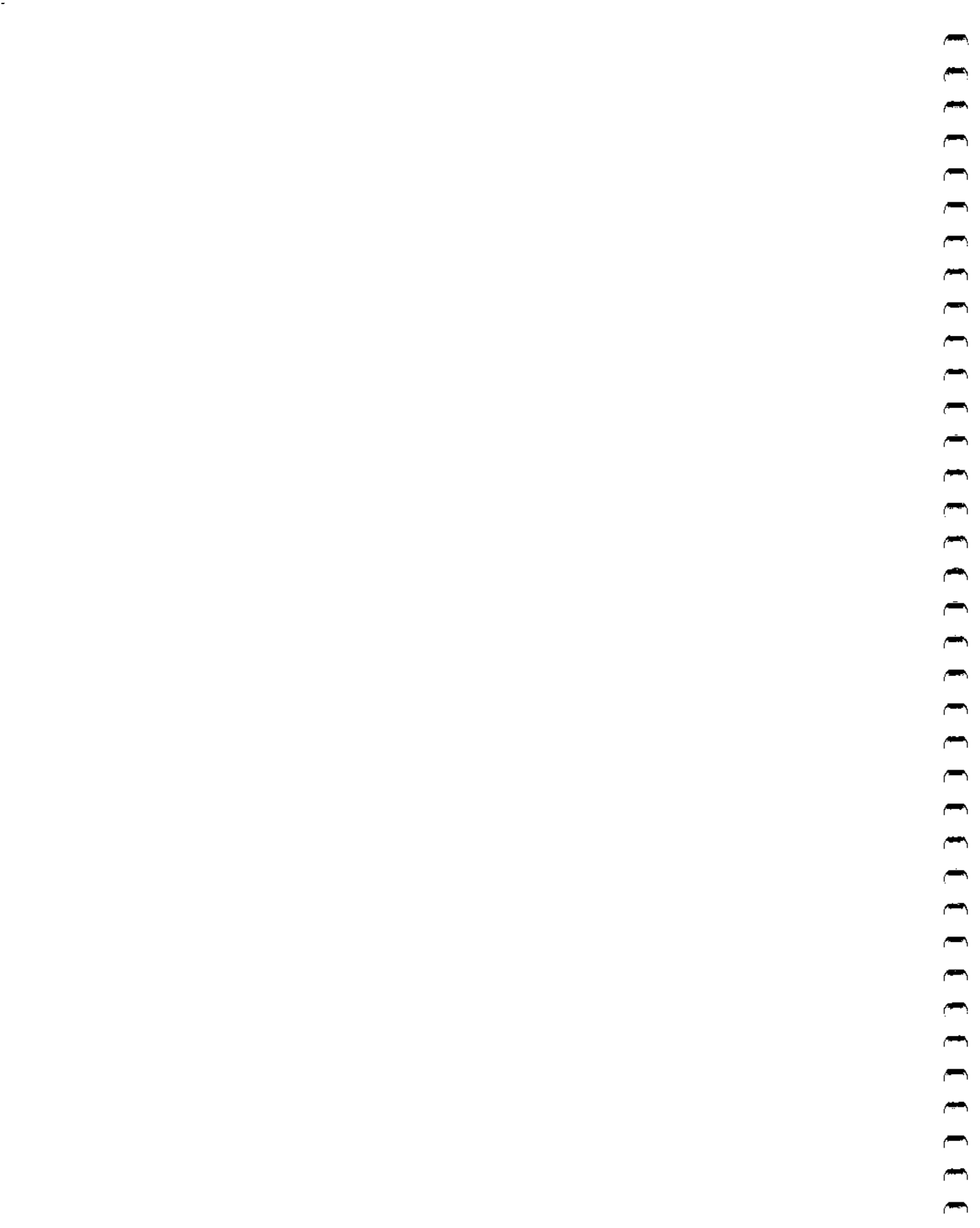
If any file you specify doesn’t exist, the file is created unless you specify the **-c** option. If no time is specified, the current time is used. The **-a** option changes only the access time of the file; the **-m** option changes only the modification time of the file.

If you don’t specify any options, **touch** behaves as if the **-a** and **-m** options were chosen.

Environment variables: **TZ** Specifies the local time for the specified time zone.

Exit status: 0 Successful completion.
 >0 An error occurred.

See also: `chmod`, `ls`



Syntax: `tr [-cs] [-r filename] string1 string2`
`tr -d [-c] [-r filename] string1`

Options:

- `-c` Complement the set of characters in *string1* with respect to the universe of characters from 00 through ff hex. Characters in *string1* are copied unchanged while all other characters are translated.
- `-d` Delete all input characters in *string1* (or not in *string1* for `-dc`).
- `-s` Squeeze all output strings of one or more instances of a single character in *string2* to a single instance of that character.
- string1* Translation character string (translate from).
- string2* Translation character string (translate to).

QNX extensions: `-r filename` Translate the named file in place (don't use `stdin/stdout`).

Examples: Either of these examples converts all lowercase characters in the input to the corresponding uppercase characters:

```
tr '[:lower:]' '[:upper:]' <file1 >file2
tr '[a-z]' '[A-Z]' <file1 >file2
```

The following example creates a list of all words in **file1** one per line in **file2** where a word is taken to be a maximal string of letters. (Octal 012 is the code for **<newline>**).

```
tr -cs "[:alpha:]" "[\012*]" <file 1 >file2
```

The next example converts a DOS file into a QNX file:

```
tr [.\15 \12 .] \12 <infile >outfile
```

Description: The **tr** utility copies the standard input to the standard output with substitution or deletion of selected characters. The options specified and the *string1* and *string2* operands control translations that occur while copying characters.

The default behavior is to replace each input character found in *string1* with the character at the same position in *string2*, while copying characters not in *string1* unchanged.

When *string2* is shorter than *string1*, *string2* is extended to the length of *string1* by duplicating the last character of *string2*. If *string2* is explicitly a string of zero length, it is padded with NUL characters (i.e. "").



The *string1* and *string2* operands often require quoting to avoid interpretation by the shell. Single quotes are usually the proper quoting mechanism.

Conventions for *string1* and *string2*

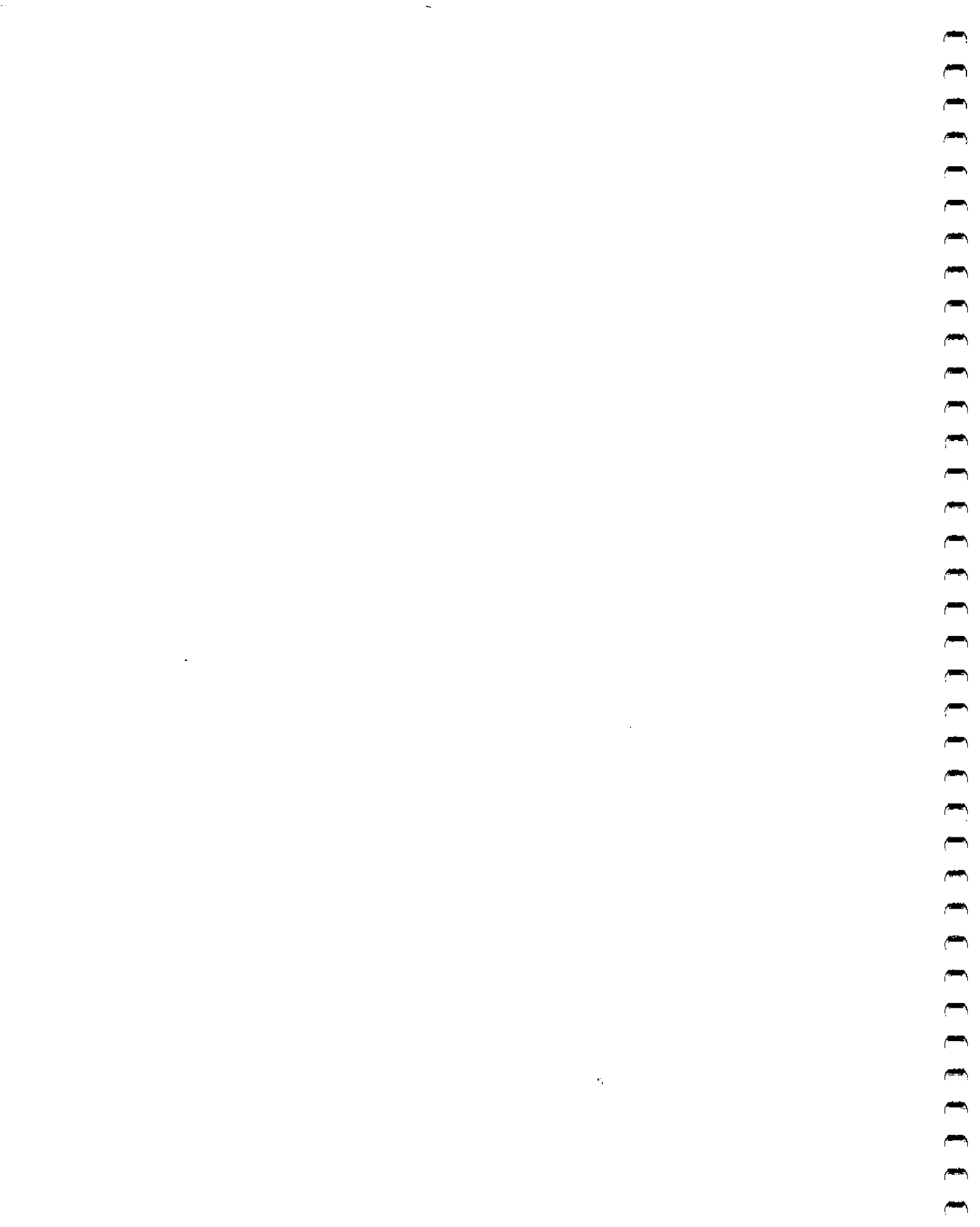
The following conventions can be used in *string1* or *string2* or both to specify characters, character ranges, character classes, or collating elements:

<i>character</i>	Represents that character.
<i>\octal</i>	A backslash followed by 1, 2, or 3 octal digits represents a character with that encoded value.
<i>\character</i>	A backslash followed by any character except an octal digit represents that character.
<i>[c-c]</i>	Represents the range of characters between the range endpoints, inclusive.
<i>[[:class:]]</i>	Represents all characters belonging to the defined character class. Allowable names for <i>class</i> are: alpha upper lower digit xdigit alnum space punct print graph cntrl
<i>[[.cs.]]</i>	Represents a collating symbol. Multicharacter collating symbols must be represented as collating symbols to distinguish them from a string of the same characters.

[*x*n*]** Represents *n* repeated occurrences of the character or collating symbol *x*. This expression is valid only in *string2*. If *n* is omitted or is zero, it is interpreted as large enough to extend the *string2*-based sequence to the length of the *string1*-based sequence. If *n* has a leading zero, it is interpreted as an octal value. Otherwise, it is interpreted as a decimal value.

Exit status: 0 All input was processed successfully.
1 An error occurred.

See also: **awk, sed**



Syntax: `umask [-o|-s|mask]`

Options:

- `-o` Display the current mask, in octal.
- `-s` Display the current mask in symbolic form. This is the default output.
- `mask` The file mode creation *mask* can be specified either as an octal number or as a symbolic representation.

If the *mask* is specified in octal form, it replaces the current file mode creation mask. Every bit that is set describes a mode bit that *won't* be allowed in the file mode of created files. In other words, it says: “mask this bit off.”

The symbolic form of the *mask* is a symbolic expression that modifies or replaces the current file mode creation mask. The form of the symbolic *mask* is similar to that of the mode operand for the **chmod** utility:

`[[augo] [+|-|=] [rwx]] [,symbolic_mask]`

- a** user, group and other access
- u** user access
- g** group access
- o** other access
- +** add these permissions to the current mask
- remove these permissions from the current mask
- =** replace the current mask with these permissions
- r** read permission
- w** write permission
- x** execute permission

Once the symbolic mask expression has been applied to the current file mode creation mask, any occurrence of **[r,w,x]** describes a mode bit that *will* be allowed in the file mode of created files. The absence of a symbol

umask

get or set the file mode creation mask (POSIX)

means that permission is not allowed and is masked “off.”

- Examples:**
- 1) Set mask to allow read, write, and execute, for user, group, and others:

```
umask a=rwx
```

Display the current file creation mode mask in symbolic form:

```
umask -s
```

```
u=rwx,g=rwx,o=rwx      allow all permissions
```

Display the current file creation mode mask in octal:

```
umask -o
```

```
00                      no masked permissions
```

- 2) Specify that no permissions for group and others be allowed; set the mask to allow only read and write for user only:

```
umask u=rw
```

Display the current file creation mode mask in symbolic form:

```
umask
```

```
u=rw,g=,o=             allow only these permissions
```

Display the current file creation mode mask in octal:

```
umask -o
```

```
0177                   mask off these permissions
```

- 3) Add read permissions for group and others:

```
umask go+r
```

Display the current file creation mode mask in symbolic form:

```
umask
```

```
u=rw,g=r,o=r          allow only these permissions
```

Display the current file creation mode mask in octal:

```
umask -o
```

```
0133
```

mask off these permissions

Description:

The **umask** utility sets the file mode creation mask of the invoking process to the value specified by the *mask* operand. The file mode creation mask affects the initial value of the file permission bits of subsequently created files when no mode is specified.

When files are created without specifying the permission mode bits, the filesystem assigns default permissions of 0777 (rwxrwxrwx) to directories and executable files, thereby giving read, write, and execute privileges for user, group, and others. For files that are not executable, permissions of 0666 (rw-rw-rw-) are assigned. The **umask** utility is used to adjust these defaults.

The file mode creation mask is inherited by any children of the current process.

Either of the display forms (**-o** or **-s**) can be used as the *mask* operand to a subsequent invocation of the **umask** utility.

As in the **chmod** utility, the use of the octal number form of mask values is deprecated.

Exit status:

0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

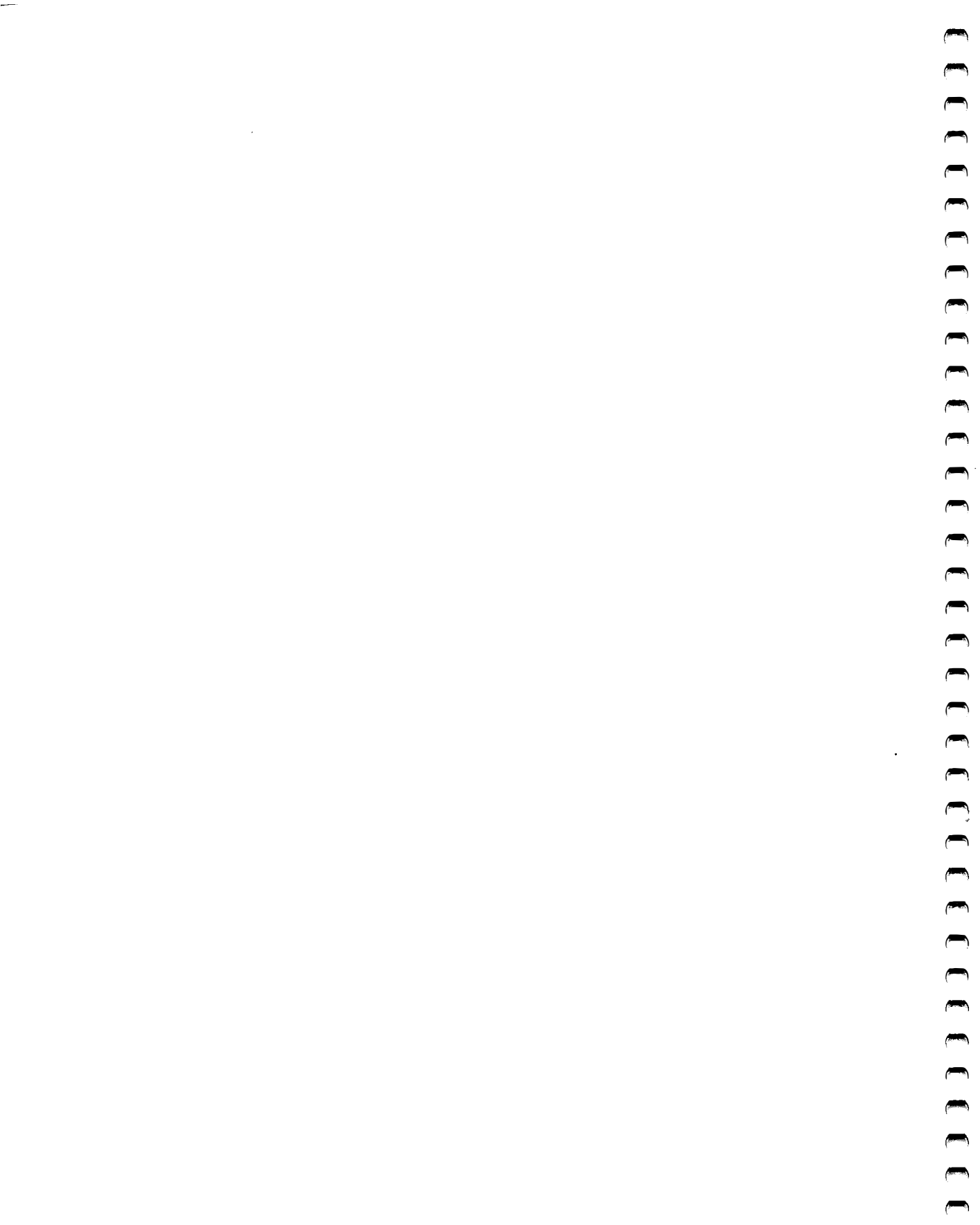
>0 An error occurred.

Consequences of errors:

If any error occurs, the value of the process's file mode creation mask is left unmodified.

See also:

chmod



Syntax: **use** *utilityname*

utilityname The utility for which a usage message will be displayed.

Examples: Print a usage message for the **ls** utility:

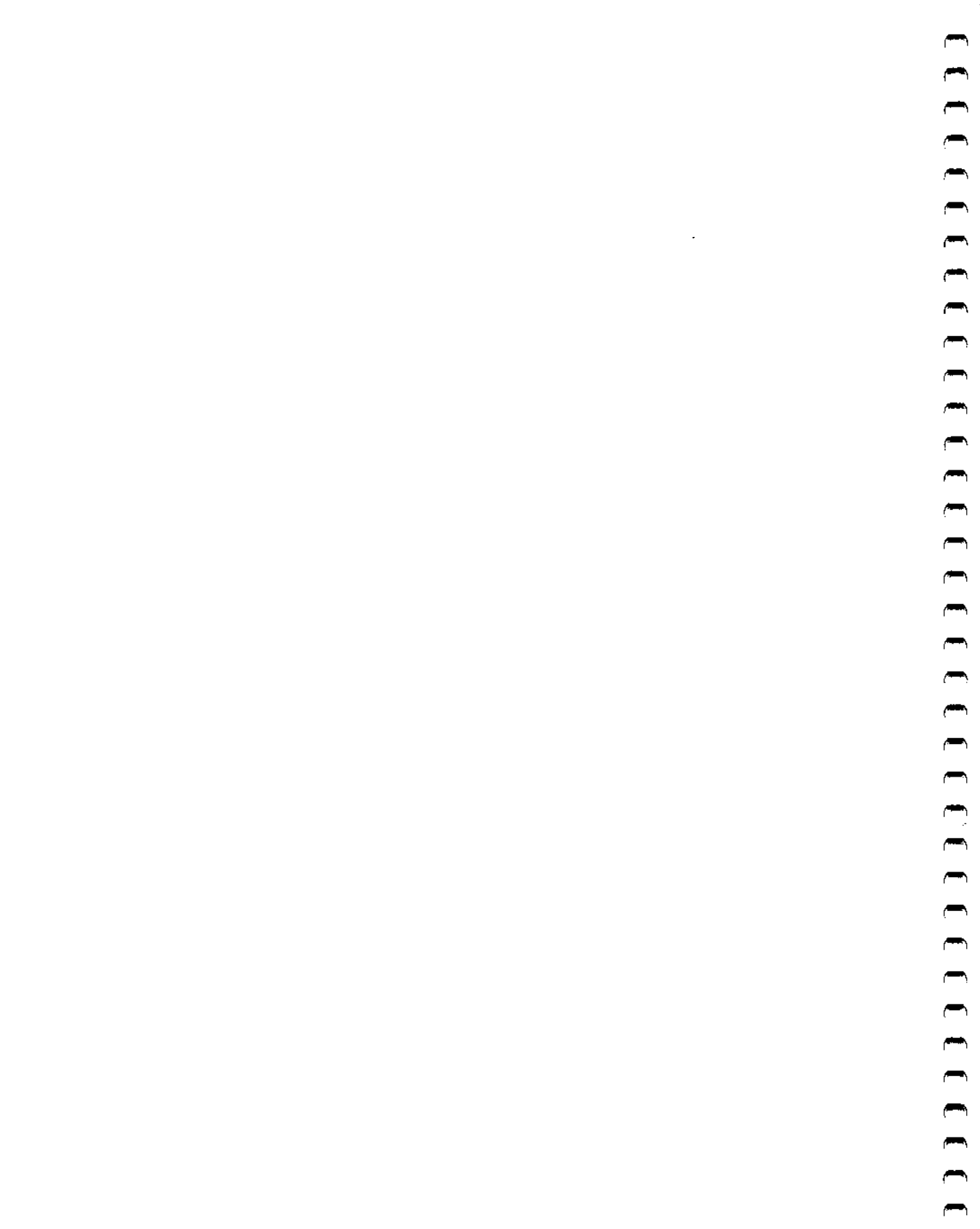
use ls

Description: The **use** utility prints a usage message for the specified executable program.

Usage messages are implemented using a special form of resource record in the load modules. The **use** utility will search for *utilityname* using the default command search (**PATH**) and will print the usage message it finds contained in the load file.

If the **LANG** environment variable is set, a usage message of that language will be displayed if it is available.

See also: **usemsg**



Syntax: **usemsg** [-c] *loadfile* [*msgfile*]

Options: -c The usage message is contained in a C source program delimited by:

```
#ifdef __USAGE
...
#endif
```

Note that there are *two* underscores before **USAGE**.

loadfile The name of an executable program to extract or insert a usage message. The current **PATH** environment variable is searched to locate *loadfile*.

msgfile A text file or a C source file containing a usage message (see -c). If the *msgfile* name ends in .c, then a C source is assumed. If present, this argument will be used as the name of the message file to insert into the load file. If the *msgfile* argument is not present, the usage message is read from the load file and printed to standard output.

Examples: Insert usage message from C source into **myprog**:

```
usemsg myprog myprog.c
```

Extract the entire usage message for **sinit**, edit the message, then reinsert the changed message:

```
usemsg sinit >sinitmsg
```

```
vedit sinitmsg
```

```
usemsg sinit sinitmsg
```

Description: The **usemsg** utility lets you examine or change the usage record contained within a QNX executable program. All utilities supplied by Quantum are shipped with a usage message describing their options. This information is kept in a resource record in the load file. Since this usage text is not loaded into memory when the program is run, you can make it as large as 32K characters without affecting the size of your program at runtime.

usemsg

change usage message for a command (QNX)

Users will use the **use** utility to print usage messages. For example:

```
use ls
use more
use sin
```

Developers will want to use the **usemsg** utility to add usage messages to their programs. There are two forms of adding a usage message to a load file. One form assumes a simple text file while the other assumes that the usage message is contained in a C source program:

```
usemsg program textfile
usemsg program program.c
```

In the second form the C source will be scanned and all text between an **#ifdef __USAGE** and the next **#endif** will be used. In both cases any existing usage message is replaced by the new message. Note that this utility lets you both change existing usage messages and add usage messages to programs that do not contain them. You do not need the program source.

The **usemsg** utility provides a simple grammar that allows it to support usage messages in several different languages. It also supports different messages linked to the name used to invoke the usage. For example, if **less** and **more** were links to the same load file they could each have their own usage within the same usage record within the file.

The grammar consists of the special symbol **%** in the first column followed by an action character as follows:

%%	a single %
%-command	the start of a specific command
%=language	the start of a new language
%C <tab>	replace with name of command and a space
<tab>	insert spaces equal to the length of the command + 1

To extract the entire usage message, including all languages and the grammar control sequences, you name the *loadfile* and don't specify a *msgfile*.

The *%-command* and *%=language* are both optional. If both are specified then the *%-command* would be followed by one or more *%=languages* followed by another *%-command* and another set of *%=languages*. The following examples should clarify the required nesting:

%C a single language message

%=english

%C an English language message

%=french

%C a French language message

%-less

%C single language message for less

%-more

%C single language message for more

%-less

%=english

%C an English language message for less

%=french

%C a French language message for less

%-more

%=english

%C an English language message for more

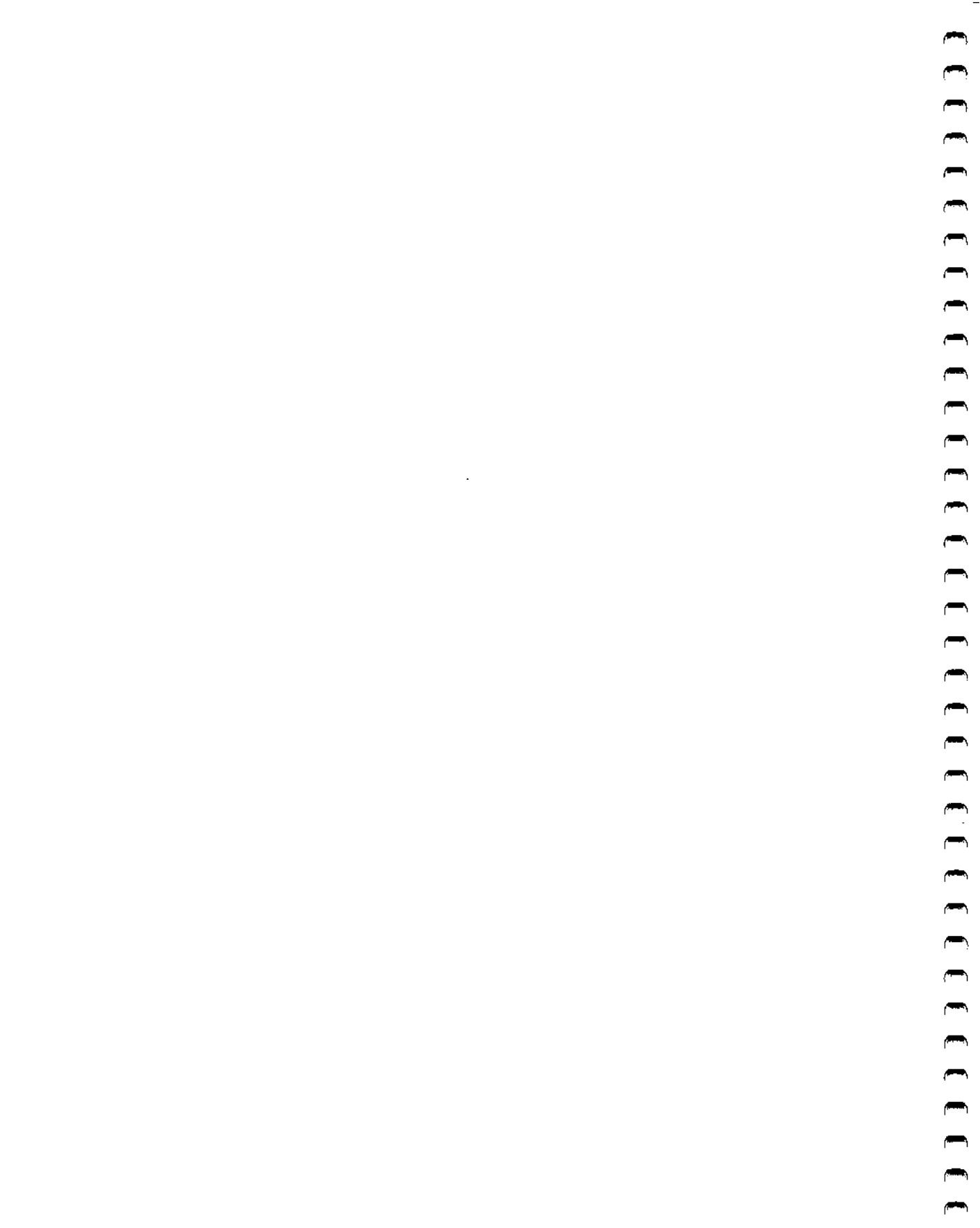
%=french

%C a French language message for more

Users will use the **use** utility to print the usage message.

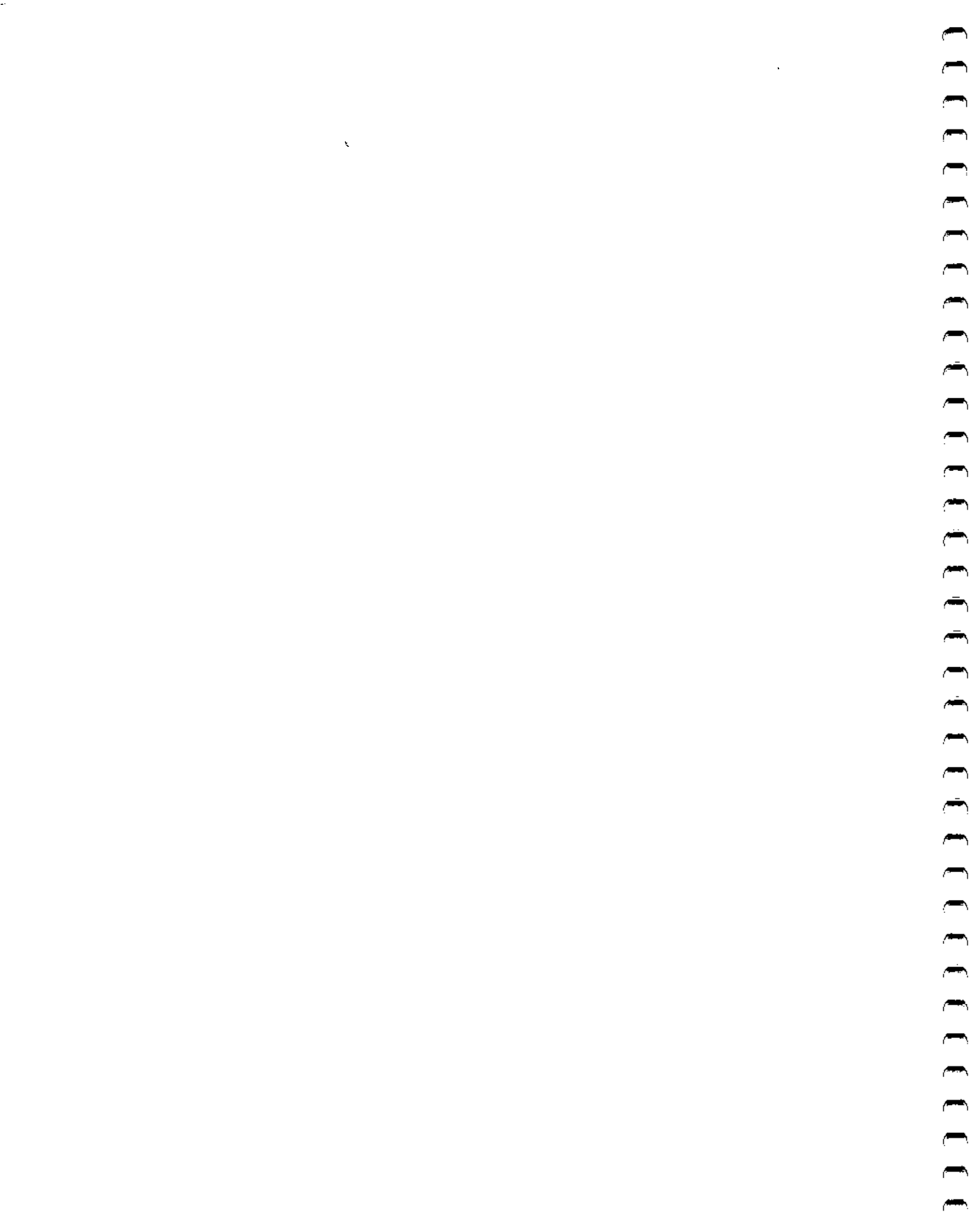
If multiple language usage messages are available, then **use** employs the **LANG** environment variable to select a language. If **LANG** isn't defined or doesn't match any language present, then the first usage message is printed. Likewise, if multiple command names are present, the command passed as an argument is used to select a command. If no match occurs, the first usage message is printed.

See also: **use**



vedit [*file*]

Description: This editor is fully described in the *VEDIT Text Editor* manual.



Syntax: `wc [-clw] [-ht] [file...]`

Options:

- `-c` Write to the standard output the number of bytes in each input file.
- `-l` Write to the standard output the number of lines in each input file.
- `-w` Write to the standard output the number of words in each input file.
- file* The pathname of an input file. If no files are specified, the standard input is used.

QNX extensions:

- `-h` Display headers.
- `-t` Don't display totals.

Description: For each input file, the **wc** utility counts lines, words, and characters (bytes) in the file and writes the results to the standard output.

If you don't specify any options, **wc** writes counts of lines, words and characters, in that order. If you specify one or more of the options `-l`, `-w`, or `-c`, the **wc** utility reports only the information selected for the specified options. The order in which you specify the options `-l`, `-w` and `-c` determines the order in which the number of lines, words, or bytes are written.

If you specify more than one input file, **wc** also writes a total count for all named files, for each option.

The utility considers a *word* to be a string of characters delimited by white space.

To get its line count, **wc** counts **<newline>** characters. If the last line of a file lacks the **<newline>** terminator, it isn't counted.

WC

word, line, and byte count (POSIX)

Exit status: 0 Successful completion.
 >0 An error occurred.

Files: The input files are text files.

See also: `cmp`, `diff`

Syntax: **which** [-**alf**] *program...*

Options:

- a** Find all occurrences of *program* in **PATH**
- f** Display the full QNX pathname
- l** Display long format (as in **ls -l**) for each program found

Examples: Display full pathname (includes node) and long status for all versions of the **ls** utility found in **PATH**:

```
which -alf ls
```

Display pathname for the **which** utility:

```
which which
```

Description: The **which** utility searches the filesystem for each *program* specified. The user's current **PATH** environment variable is used as the search path.

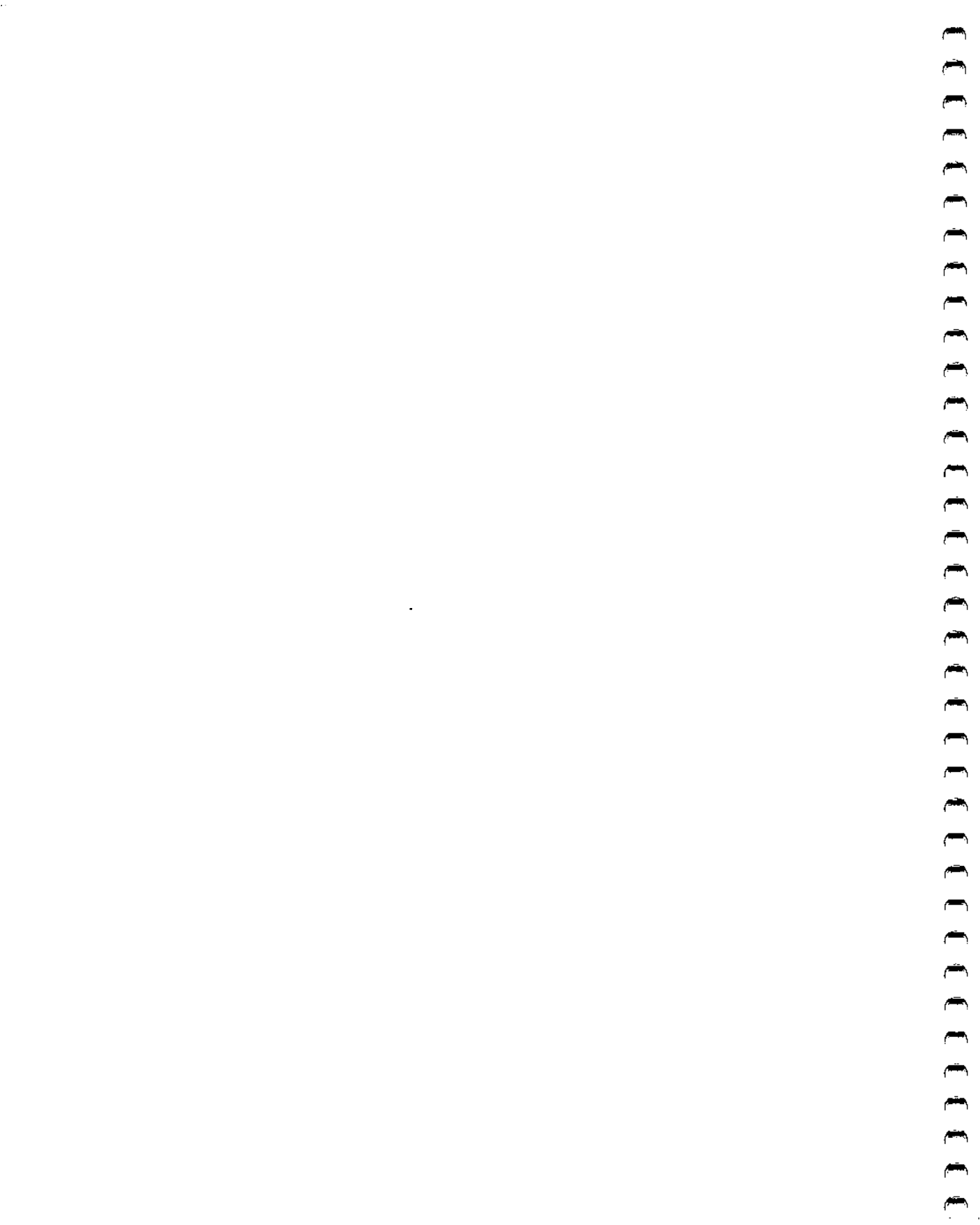
Exit status:

- 0 All input files were searched successfully.
- >0 An error occurred.

Environment variables:

PATH A colon-separated list of directories to search for QNX executables. The default **PATH** is:
 :**/bin:/usr/bin**

See also: **sh**



Syntax: **who** [-aimow] [-n *node*]

- Options:**
- a** Get information on all users on the network.
 - i** Output the user's input "idle time" in addition to any other information. The idle time is the time since any input occurred on the user's terminal. This is the default option.
 - m** (me) Output only information about the current terminal.
 - n** *node*
Get **who** information from the specified *node*. The default is the local node.
 - o** Output the user's output "idle time" in addition to any other information. The output idle time is the time since any output occurred to the user's terminal.
 - w** Show the state of each terminal in the **who** output as the + character if the terminal allows write access to other users and the - character if such access is denied. The ? character is printed if the information can't, for some reason, be determined.

Examples: Display users on the current node:

who

Display information about me:

who -m

Display all users on the network:

who -a

Description: The **who** utility lists various pieces of information about any accessible users.

If you don't specify any options, the default output includes the user's login name, terminal name, time at which the user logged in, input idle time, and the command being run.

If an idle time is reported as dashes, then no input (output) occurred.

who

display who is on the system (QNX)

Exit status: 0 Successful completion.
 >0 An error occurred.

See also: `login`

Caveats: When reporting on the command being executed, the `who` utility assumes the deepest program in the process tree with the same process group number as the session leader.

Syntax: **write** [-a] [-n *node*] *username* [*terminal*]

Options: **-a** Search all nodes for *username*.
 -n *node* Search only the specified *node* for *username*.

 username The user to write a message to.
 terminal The terminal to write a message to.

Examples: Write to Eric on the local node:

```
write eric
```

Search the network for the user **dtddodge** and write to his terminal or console:

```
write -a dtddodge
```

Write to **aboyd** on node 10:

```
write -n 10 aboyd
```

Write to Steve on **//60/dev/ser1**:

```
write steve //60/dev/ser1
```

Description: The **write** utility is used to send a message to the terminal or console of another user.

When **write** is invoked, it writes the message:

```
Message from sender-login-id (sending-terminal) [date]
```

to the recipient's terminal. Then it copies its **stdin** to the recipient's terminal on a line-by-line basis until the user types <Ctrl>-<D>.

The recipient can reply (thus engaging the sender in an interactive conversation) by entering:

```
write sender-login-id sending-terminal
```

using the information displayed in the message header written by the sender to his screen.

write

write a message to a user's terminal (UNIX)

If the first input character typed on a line is the exclamation mark (!), the **write** utility will pass the remainder of the line to the shell for execution.

Non-printable characters cannot be sent through the **write** utility. They will be expanded to octal before being written to the recipient's terminal.

When writing to a user who is logged in more than once, **write** will select one of the terminals to write to. The terminal argument can be used to force **write** to display to a particular terminal.

Unless **-n node** or **-a** is specified, **write** will only look on the sender's node for the recipient. A different node can be explicitly specified by using **-n node**, or the entire network can be searched by specifying the **-a** option.

Environment variables:

SHELL Used to determine the preferred command line interpreter for use in command escapes. If no **SHELL** environment variable is set, the **sh** utility will be used.

Exit status:

0 Successful completion.
>0 An error occurred.

See also:

who

Syntax: **ws** [*options*] [*directory* ["*command...*"]]

- Options:**
- a** *date/time* Select only those files that were last modified **after** the specified *date/time*, where *date/time* is
 " [UTC] *dd/mm/yy HH:MM:SS*"
 24 hour time (the time portion is optional). To use Coordinated Universal Time, specify **UTC**.
 If you're running **ws** from the shell, and you want to specify both the date and the time, you must use the quotation marks. If you want to specify only the date (*dd/mm/yy*), quotation marks aren't needed.
 - b** *date/time* Select only files last modified **before** the specified time (see option **-a** *date/time*).
 - c** *d=delete_char* This character, when present in *command*, deletes the previous character of the expanded string. The default is a backquote character (`).
 - c** *p=path_char* This character, when present in *command*, is replaced by the current pathname. If the *path_char* is followed by a question mark, the two-character sequence is replaced with input prompted for from the standard input. If the *path_char* is followed by another *path_char*, only the filename and not the complete pathname will be replaced. This is sometimes useful when **-r** (relative) isn't specified but you still want the relative name. The default for *path_char* is an at-sign (@).
 - d** Select directories rather than files.

-e *ignore|prompt|abort*

When a non-zero return code comes back from *command*, ignore the error, prompt to continue, or abort (exit **ws**).

-g *group[,group]...*

Indicates that only files whose group matches one of the specified groups will be selected. Groups may be specified as group numbers or group names.

-i Be interactive; before executing each command, issue a prompt requesting permission to execute. Unless the silent option is set, the entire command line is prompted; otherwise only the selected pathname is printed. A single **y** permits execution while anything else skips to the next pathname.

-l *levels*

(the lowercase letter “e”) Specifies how many *levels* down from the indicated directory will be searched. The default is **l=0** and will cause only files at the current level to be searched; no subdirectories are searched.

-m *[!]mode[,![!]mode]...*

Select only files whose mode matches one of the specified *modes*. The format for *mode* is similar to that used by the **chmod** utility—an octal number or symbolic mode of the form:

[[agou] [=] [rswx]] [, symbolic_mode]

The options of the symbolic form are:

- a** user, group, and other access
- g** group access
- o** other access
- u** user access
- r** read permission
- s** set user ID or group ID when executed
- w** write permission
- x** execute permission

A file with a mode that is a superset of or equal to one of the modes specified will be selected. If you specify a *mode* with a preceding exclamation mark (!), files matching that *mode* won't be selected.

-P [!] *pat* [, [!] *pat*] ...

Select only files matching these patterns. If you specify a *pat* with a preceding exclamation mark (!), files matching that *pat* won't be selected. (For details, see the section on "Pattern matching special characters" in the **cp** utility.)

-r (relative) The **ws** command usually generates complete pathnames from the indicated directory. But if you specify this option, **ws** executes a **cd** command when it moves to each new directory; the pathname will be the simple filename in the new directory.

-s Suppress the printing of each command before execution.

-t Ignore date/time of files being processed. By default, **ws** skips over any file that it believes was created *after* the utility started running. It does this by remembering the time it started to execute and comparing that time with the file creation times of the files it processes. If the system time has been set back from the time on the system when the file was created, **ws** will skip the file even though it was "already there." The **-t** option may be used to get around this default behavior. Note that with **-t**, the following will produce an infinite loop:

```
ws . "cp @ @.x"
```

-u *owner* [, *owner*] ...

Indicates that only files whose file owner matches one of the specified members will be selected. The owner may be specified by user ID or username.

Examples: Remove all files at current directory:

```
ws "rm @"
```

Remove all files in Bruce's home directory:

```
ws -l 20 /usr/bruce "rm @"
```

Remove all files on the hard disk owned by Bruce:

```
ws -l 100 -u bruce / "rm @"
```

Display info on all the directories under /usr/bruce:

```
ws -dl20 /usr/bruce "ls -ld @"
```

Move all files in the current directory matching ***data*** to a new location that will be prompted for each file:

```
ws -P \*data\* "mv @ @?"
```

Edit all C programs that were changed since Nov. 11, 1991:

```
ws -a 11/11/91 -P \*.c "vedit @"
```

Search for the **xyz** file; if found, print its complete pathname:

```
ws -g -Pxyz -l12 / "echo @"
```

Description:

The **ws** utility increases the power and scope of existing QNX commands. Study it carefully—it's a very powerful command. The functionality of **ws** overlaps that of the **find** utility, which is found in a wide variety of systems; **ws**, however, is strictly QNX in origin.

The simplest form of the command is:

```
ws "command arguments"
```

which executes the command in double quotes on each file in the current directory. The double quotes are necessary.

Any occurrence of the at-sign character (@) is replaced by the current filename. For example:

<pre>ws "echo @"</pre>	display the name of each file in the current directory
<pre>ws "rm -f @"</pre>	remove each file in the current directory

You may specify a number of options after the command. The two you'll probably use most are **-i** and **-s**. The interactive option (**-i**) prompts you for each possible execution of the command. A single **y** executes the command, anything else skips execution and proceeds to the next. The silent option (**-s**) suppresses the printing of each command before execution.

The **ws** utility doesn't normally descend to subdirectories in the current directory. You can override this default behavior by specifying the number of directories to descend using the **-l levels** option. The following example releases all files under the directory **/usr/joe** including all subdirectories. The directories are then removed. The two commands effectively release all a user's files, perhaps in preparation of removing their user ID.

```
ws -l 30 /usr/joe "rm -f @" Release all files
ws -dl 30 /usr/joe "rmdir @" Release directories
```

Note that in this case a particular directory was specified, and in the second example the **-d** option matched directories, **not** files. Note that this is just an example of how **ws** could be used. The following command:

```
rm -Rf /usr/joe
```

is a better choice for this particular task.

The **-P [!]pattern** option follows the same rules as used by the **cp** utility. Both use the *fnmatch()* C function to match patterns. (For details, see the section on "Pattern matching special characters" in the **cp** utility.)

The at-sign prompt (**@?**) lets you increase the flexibility of the **ws** command. There's no limit, within reason, to the number of **@?** prompts you may have on a command line. The following example allows a superuser to interactively set the permissions on each user directory:

```
ws -id /usr "chmod @? @"
```

The relative (**-r**) option causes **ws** to issue a **cd** command before entering each subdirectory. The pathnames expanded by the at-sign (**@**) will contain the last component of the full pathname. For example:

```
ws -l3 1:/ "echo @"           display complete pathnames
```

AND

```
ws -rl3 10 1:/ "echo @"       display simple filenames
```

All examples above have contained a *command* and some have contained an overriding *directory* as their first argument. If you specify neither, **ws** operates on the current directory and prompts you for command lines from the standard input. Each input line will be executed on all selected files and the user will then be prompted for another command. An empty line will terminate the command.

If you specify a *directory*, the syntax requires that you enter a *command*. If the command is an empty string, you'll be prompted. For example:

```
ws                             prompt for commands and operate on  
                                current directory
```

```
ws /usr/fred ""                prompt and operate on /usr/fred
```

You can use the **ws** utility as a filter to which a series of commands is piped. The following example assumes that the file **my_bin** contains a list of commands you wish to have executed on all files ending in **.c** at the current directory:

```
ws -P\*.c <my_bin
```

Exit status: 0 No errors were encountered.
 >0 An error occurred.

See also: **cp, find, xargs**

Caveats: The **ws** utility will not operate on any file that has a file creation date later than the time at which **ws** started running.

Syntax: **xargs** **[-n *number*]** **[-s *size*]** **[-t]**
 [-i] [*program* [*initial-arguments*]]

Options:

- n *number*** Every time *number* strings are read, execute *program*. The default for *number* is 255.
- s *size*** Set the maximum command buffer to *size* characters, including *program* and *initial-argument*. The default for *size* is 4096.
- t** Trace. Print each *program* on standard error before executing.
- program* The name of the program to be executed. The program must be found by searching the path using the **PATH** environment variable. If *program* is omitted, the default is the **echo** utility.
- initial-arguments* One or more arguments to *program* that are presented every time *program* is executed.

QNX extensions:

- i** Execute in “insert mode” —the *program* will be executed **once** for each item in standard input. Replaces each occurrence of { } in *initial-arguments* with the argument read from standard input. If there are no occurrences of { } in *initial-arguments*, the argument is appended to the initial list.

Examples: Use **cmp** to determine whether the files in the directory **old_data** are the same as the files in the directory **new_data**:

```
ls old_data | xargs cmp old_data/{} new_data/{}>
```

Display the files in the current working directory, and all subdirectories, in two columns:

```
find . -print | xargs -n 2 echo>
```


xargs

construct argument list(s) and execute program (POSIX)

Description: The **xargs** utility uses character strings, read from standard input, to construct a command line which it executes. The specified “program” and “initial-arguments” are placed at the beginning of the command line, followed by some number of character strings read. This process continues until the end of the file.

The utility executes a given program with *initial-arguments* one or more times using the parameters read from standard input. The number of strings appended may be limited by the **-n** option; the size of the command line may be limited by the **-s** option.

The strings are separated by <blank>s or <newline>s, which may be embedded in the strings by prefixing them with \ or enclosing them in quotes ("). To use the quote character as itself, it must be prefixed with a \.

The **-i** option causes the command to be executed for each string read. Instead of the normal process of appending the string to the command buffer, the *initial-arguments* are scanned, and every occurrence of { } is replaced by the string. If { } doesn't occur in *initial-arguments*, the string is appended to the command line and executed.

When a program is executed, it inherits standard output and standard error from **xargs**. Standard input is set to the controlling tty.

The **xargs** utility will always limit the total command buffer size to 4096 characters. The following example may be used to verify the integrity of data files on a floppy disk (mounted as **/fd**).

```
find /fd -print | xargs cksum | diff check_file -
```

In this example, **find** will print out the names of each file on the mounted filesystem. The **xargs** utility will group the filenames up for **cksum**, to minimize the number of times **cksum** must be executed. The **diff** utility is then used to verify that the calculated checksums are the same as recorded in the file **check_file**.

It is important to note that the following command:

```
find /fd -exec cksum {} \; | diff check_file -
```

would perform the same utility, but would require **cksum** to be reloaded, once for each file in **/fd**.

Exit status: 0 All invocations of *program* executed successfully.
 >0 An error occurred.

See also: **find**



Syntax: `yacc [-dltv] [-b prefix] file`

Options:

- `-d` Write a header file containing manifests for all tokens.
- `-l` The file does not contain any *#line* constructs. If this option is not specified, then **yacc** will attempt to adjust any such constructs as necessary.
- `-t` Generate a parser with debugging information, which may be enabled by setting **yydebug** to 1.
- `-v` Be verbose; generate statistics of the parser.
- `-b prefix` Use *prefix* rather than **y** for the generated filename prefixes:
 - `y.tab.h`
 - `y.tab.c`
 - `y.output`

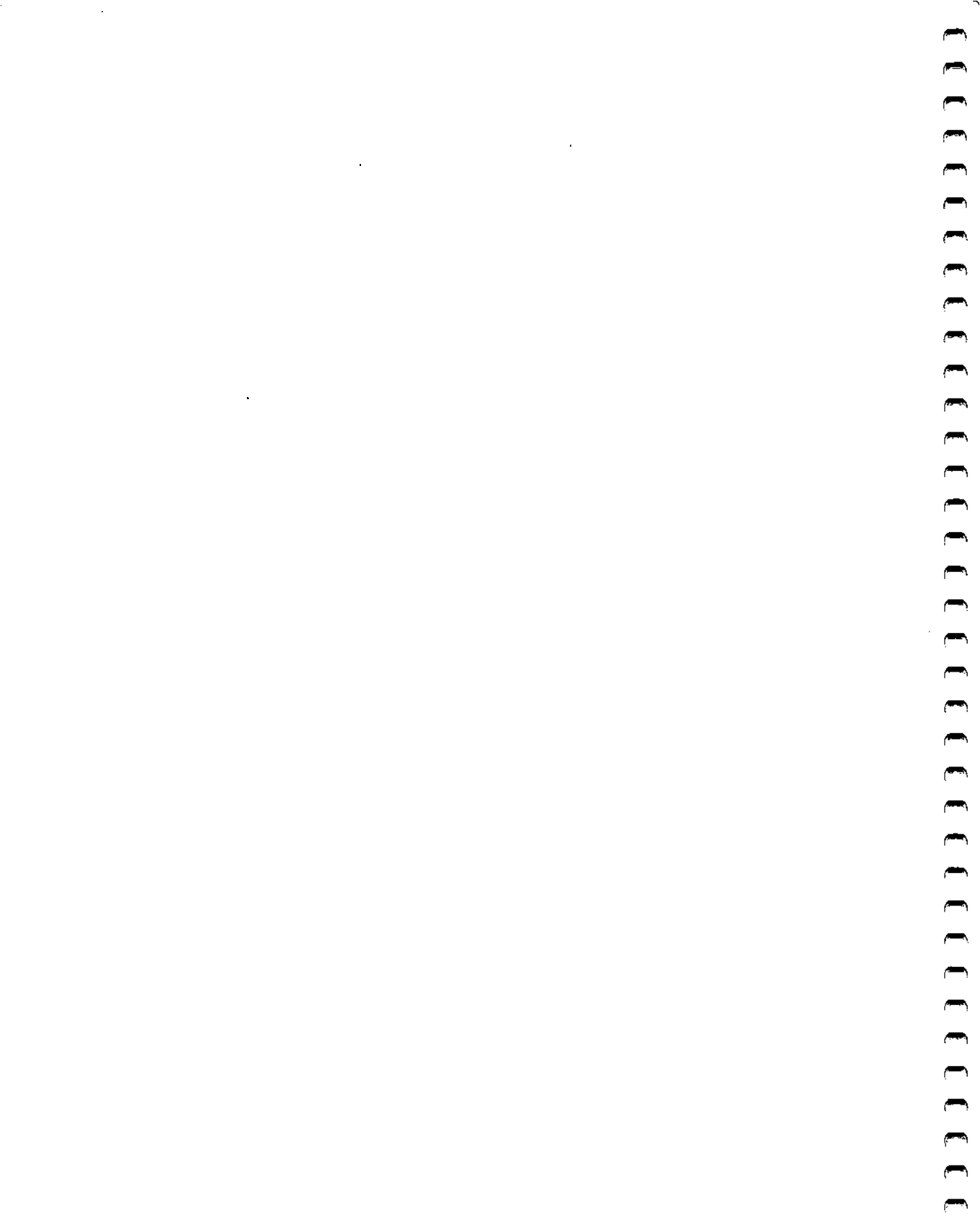
Description: The **yacc** utility transforms a specification of a context-free grammar into a C language function that implements an LR(1) parsing algorithm. Ambiguities in the grammar may be resolved using precedence rules within the specification. The C language function is named `yyparse()`, and calls `yylex()` to get input tokens, which are integer values. The value 0 is considered special, as being end of input.

See also: **lex**

A.V. Aho and S.C Johnson, "LR Parsing," *Computing Surveys*, June 1974

S.C. Johnson, "Yacc—Yet Another Compiler-Compiler," *Bell Laboratories Computing Science Technical Report # 32*, July 1978

Brian W. Kernighan and Rob Pike, *The UNIX Programming Environment*, Prentice-Hall, 1984



Syntax: **zap** [-Rpv] *file*

zap [-Rpv] [-l|-u] *directory*

Options: **-R** Enter Rambo mode! Override the filesystem (allow **zap** when files are open on the disk).

-l (lowercase “el”) List previously zapped files in the *directory*.

-p Pause before starting (for floppy disks).

-u “Unzap” files in the *directory*.

-v Be verbose; report results.

file The name of the file to remove.

directory The name of the directory containing previously zapped files.

Examples: **zap** junk
 zap /usr/bill/junk

Description: You should use **zap** on a file if:

- the file is known to contain bad disk blocks

 OR

- you have been instructed to do so by **chkfsys**

The **zap** utility releases a file by clearing the directory entry for that file. The disk blocks used by the file aren’t reclaimed. Therefore, if you use **zap** repeatedly, you’ll reduce the total number of disk blocks available on the disk. You can reclaim these, however, by running the **chkfsys** utility when the system is idle.

Normally, you should use the **rm** or **rmdir** commands to release files or directories.

Previously zapped files may be listed in any specified directory using the **-l** option.

zap

destroy a damaged file (QNX)

You can “unzap” or recover zapped files by using the **-u** option. The utility will prompt for each file that was zapped in the specified directory.

Exit status: 0 Successful.
 >0 An error occurred.

See also: **chkfsys, rm, rmdir**

Caveats: To remove a file from a device that has any files open, you must use the **-R** (Rambo) option.

