GNU `Core-utils`

A set of command line utilities
for version 4.5.4, 2 December 2002

David MacKenzie et al.

This manual documents version 4.5.4 of the GNU command line utilities.

1 Introduction

This manual is a work in progress: many sections make no attempt to explain basic concepts in a way suitable for novices. Thus, if you are interested, please get involved in improving this manual. The entire GNU community will benefit.

The GNU utilities documented here are mostly compatible with the POSIX standard. Please report bugs to `bug-coreutils@gnu.org`. Remember to include the version number, machine architecture, input files, and any other information needed to reproduce the bug: your input, what you expected, what you got, and why it is wrong. Diffs are welcome, but please include a description of the problem as well, since this is sometimes difficult to infer. See section "Bugs" in *Using and Porting GNU CC*.

This manual was originally derived from the Unix man pages in the distributions, which were written by David MacKenzie and updated by Jim Meyering. What you are reading now is the authoritative documentation for these utilities; the man pages are no longer being maintained. The original `fmt` man page was written by Ross Paterson. François Pinard did the initial conversion to Texinfo format. Karl Berry did the indexing, some reorganization, and editing of the results. Brian Youmans of the Free Software Foundation office staff combined the manuals for textutils, fileutils, and sh-utils to produce the present omnibus manual. Richard Stallman contributed his usual invaluable insights to the overall process.

2 Common options

Certain options are available in all of these programs. Rather than writing identical descriptions for each of the programs, they are described here. (In fact, every GNU program accepts (or should accept) these options.)

Normally options and operands can appear in any order, and programs act as if all the options appear before any operands. For example, '`sort -r passwd -t :`' acts like '`sort -r -t : passwd`', since '`:`' is an option-argument of '`-t`'. However, if the `POSIXLY_CORRECT` environment variable is set, options must appear before operands, unless otherwise specified for a particular command.

Some of these programs recognize the '`--help`' and '`--version`' options only when one of them is the sole command line argument.

'`--help`' Print a usage message listing all available options, then exit successfully.

'`--version`'
: Print the version number, then exit successfully.

'`--`' Delimit the option list. Later arguments, if any, are treated as operands even if they begin with '`-`'. For example, '`sort -- -r`' reads from the file named '`-r`'.

A single '`-`' is not really an option, though it looks like one. It stands for standard input, or for standard output if that is clear from the context, and it can be used either as an operand or as an option-argument. For example, '`sort -o - -`' outputs to standard output and reads from standard input, and is equivalent to plain '`sort`'. Unless otherwise specified, '`-`' can appear in any context that requires a file name.

2.1 Backup options

Some GNU programs (at least `cp`, `install`, `ln`, and `mv`) optionally make backups of files before writing new versions. These options control the details of these backups. The options are also briefly mentioned in the descriptions of the particular programs.

'`-b`'
'`--backup[=method]`'
: Make a backup of each file that would otherwise be overwritten or removed. Without this option, the original versions are destroyed. Use *method* to determine the type of backups to make. When this option is used but *method* is not specified, then the value of the `VERSION_CONTROL` environment variable is used. And if `VERSION_CONTROL` is not set, the default backup type is '`existing`'.

  Note that the short form of this option, '`-b`' does not accept any argument. Using '`-b`' is equivalent to using '`--backup=existing`'.

  This option corresponds to the Emacs variable '`version-control`'; the values for *method* are the same as those used in Emacs. This option also accepts more descriptive names. The valid *method*s are (unique abbreviations are accepted):

  '`none`'
  '`off`' Never make backups.

'numbered'
't' Always make numbered backups.

'existing'
'nil' Make numbered backups of files that already have them, simple backups of the others.

'simple'
'never' Always make simple backups. Please note 'never' is not to be confused with 'none'.

'-S *suffix*'
'--suffix=*suffix*'
: Append *suffix* to each backup file made with '-b'. If this option is not specified, the value of the SIMPLE_BACKUP_SUFFIX environment variable is used. And if SIMPLE_BACKUP_SUFFIX is not set, the default is '~', just as in Emacs.

'--version-control=*method*'
: This option is obsolete and will be removed in a future release. It has been replaced with `--backup`.

2.2 Block size

Some GNU programs (at least `df`, `du`, and `ls`) display sizes in "blocks". You can adjust the block size and method of display to make sizes easier to read. The block size used for display is independent of any filesystem block size. Fractional block counts are rounded up to the nearest integer.

The default block size is chosen by examining the following environment variables in turn; the first one that is set determines the block size.

DF_BLOCK_SIZE
: This specifies the default block size for the `df` command. Similarly, DU_BLOCK_SIZE specifies the default for `du` and LS_BLOCK_SIZE for `ls`.

BLOCK_SIZE
: This specifies the default block size for all three commands, if the above command-specific environment variables are not set.

POSIXLY_CORRECT
: If neither the *command*_BLOCK_SIZE nor the BLOCK_SIZE variables are set, but this variable is set, the block size defaults to 512.

If none of the above environment variables are set, the block size currently defaults to 1024 bytes in most contexts, but this number may change in the future. For `ls` file sizes, the block size defaults to 1 byte.

A block size specification can be a positive integer specifying the number of bytes per block, or it can be `human-readable` or `si` to select a human-readable format. Integers may be followed by suffixes that are upward compatible with the SI prefixes (http://www.bipm.fr/enus/3_SI/si-prefixes.html) for decimal multiples and with the IEC 60027-2 prefixes for binary multiples (http://physics.nist.gov/cuu/Units/binary.html).

With human-readable formats, output sizes are followed by a size letter such as 'M' for megabytes. `BLOCK_SIZE=human-readable` uses powers of 1024; 'M' stands for 1,048,576 bytes. `BLOCK_SIZE=si` is similar, but uses powers of 1000 and appends 'B'; 'MB' stands for 1,000,000 bytes.

A block size specification preceded by ''' causes output sizes to be displayed with thousands separators. The `LC_NUMERIC` locale specifies the thousands separator and grouping. For example, in an American English locale, '`--block-size="'1kB"`' would cause a size of 1234000 bytes to be displayed as '`1,234`'. In the default C locale, there is no thousands separator so a leading ''' has no effect.

An integer block size can be followed by a suffix to specify a multiple of that size. A bare size letter, or one followed by '`iB`', specifies a multiple using powers of 1024. A size letter followed by '`B`' specifies powers of 1000 instead. For example, '`1M`' and '`1MiB`' are equivalent to '`1048576`', whereas '`1MB`' is equivalent to '`1000000`'.

A plain suffix without a preceding integer acts as if '`1`' were prepended, except that it causes a size indication to be appended to the output. For example, '`--block-size="kB"`' displays 3000 as '`3kB`'.

The following suffixes are defined. Large sizes like `1Y` may be rejected by your computer due to limitations of its arithmetic.

'`kB`' kilobyte: $10^3 = 1000$.

'`k`'
'`K`'
'`KiB`' kibibyte: $2^1 0 = 1024$. '`K`' is special: the SI prefix is '`k`' and the IEC 60027-2 prefix is '`Ki`', but tradition and POSIX use '`k`' to mean '`KiB`'.

'`MB`' megabyte: $10^6 = 1,000,000$.

'`M`'
'`MiB`' mebibyte: $2^2 0 = 1,048,576$.

'`GB`' gigabyte: $10^9 = 1,000,000,000$.

'`G`'
'`GiB`' gibibyte: $2^3 0 = 1,073,741,824$.

'`TB`' terabyte: $10^1 2 = 1,000,000,000,000$.

'`T`'
'`TiB`' tebibyte: $2^4 0 = 1,099,511,627,776$.

'`PB`' petabyte: $10^1 5 = 1,000,000,000,000,000$.

'`P`'
'`PiB`' pebibyte: $2^5 0 = 1,125,899,906,842,624$.

'`EB`' exabyte: $10^1 8 = 1,000,000,000,000,000,000$.

'`E`'
'`EiB`' exbibyte: $2^6 0 = 1,152,921,504,606,846,976$.

'`ZB`' zettabyte: $10^2 1 = 1,000,000,000,000,000,000,000$

'Z'
'ZiB' $2^{7}0 = 1,180,591,620,717,411,303,424$. ('Zi' is a GNU extension to IEC 60027-2.)

'YB' yottabyte: $10^{2}4 = 1,000,000,000,000,000,000,000,000$.

'Y'
'YiB' $2^{8}0 = 1,208,925,819,614,629,174,706,176$. ('Yi' is a GNU extension to IEC 60027-2.)

Block size defaults can be overridden by an explicit '`--block-size=`*size*' option. The '`-k`' option is equivalent to '`--block-size=1K`', which is the default unless the `POSIXLY_CORRECT` environment variable is set. The '`-h`' or '`--human-readable`' option is equivalent to '`--block-size=human-readable`'. The '`--si`' option is equivalent to '`--block-size=si`'.

2.3 Target directory

Some GNU programs (at least `cp`, `install`, `ln`, and `mv`) allow you to specify the target directory via this option:

'`--target-directory=`*directory*'

Specify the destination *directory*.

The interface for most programs is that after processing options and a finite (possibly zero) number of fixed-position arguments, the remaining argument list is either expected to be empty, or is a list of items (usually files) that will all be handled identically. The `xargs` program is designed to work well with this convention.

The commands in the `mv`-family are unusual in that they take a variable number of arguments with a special case at the *end* (namely, the target directory). This makes it nontrivial to perform some operations, e.g., "move all files from here to ../d/", because `mv * ../d/` might exhaust the argument space, and `ls | xargs ...` doesn't have a clean way to specify an extra final argument for each invocation of the subject command. (It can be done by going through a shell command, but that requires more human labor and brain power than it should.)

The `--target-directory` option allows the `cp`, `install`, `ln`, and `mv` programs to be used conveniently with `xargs`. For example, you can move the files from the current directory to a sibling directory, `d` like this: (However, this doesn't move files whose names begin with '`.`'.)

```
ls |xargs mv --target-directory=../d
```

If you use the GNU `find` program, you can move *all* files with this command:

```
find . -mindepth 1 -maxdepth 1 \
  | xargs mv --target-directory=../d
```

But that will fail if there are no files in the current directory or if any file has a name containing a newline character. The following example removes those limitations and requires both GNU `find` and GNU `xargs`:

```
find . -mindepth 1 -maxdepth 1 -print0 \
  | xargs --null --no-run-if-empty \
```

```
mv --target-directory=../d
```

2.4 Trailing slashes

Some GNU programs (at least `cp` and `mv`) allow you to remove any trailing slashes from each *source* argument before operating on it. The `--strip-trailing-slashes` option enables this behavior.

This is useful when a *source* argument may have a trailing slash and specify a symbolic link to a directory. This scenario is in fact rather common because some shells can automatically append a trailing slash when performing file name completion on such symbolic links. Without this option, `mv`, for example, (via the system's rename function) must interpret a trailing slash as a request to dereference the symbolic link and so must rename the indirectly referenced *directory* and not the symbolic link. Although it may seem surprising that such behavior be the default, it is required by POSIX and is consistent with other parts of that standard.

2.5 Standards conformance

In a few cases, the GNU utilities' default behavior is incompatible with the POSIX standard. To suppress these incompatibilities, define the `POSIXLY_CORRECT` environment variable. Unless you are checking for POSIX conformance, you probably do not need to define `POSIXLY_CORRECT`.

Newer versions of POSIX are occasionally incompatible with older versions. For example, older versions of POSIX required the command '`sort +1`' to sort based on the second and succeeding fields in each input line, but starting with POSIX 1003.1-2001 the same command is required to sort the file named '`+1`', and you must instead use the command '`sort -k 2`' to get the field-based sort.

The GNU utilities normally conform to the version of POSIX that is standard for your system. To cause them to conform to a different version of POSIX, define the `_POSIX2_VERSION` environment variable to a value of the form *yyyymm* specifying the year and month the standard was adopted. Two values are currently supported for `_POSIX2_VERSION`: '`199209`' stands for POSIX 1003.2-1992, and '`200112`' stands for POSIX 1003.1-2001. For example, if you are running older software that assumes an older version of POSIX and uses '`sort +1`', you can work around the compatibility problems by setting '`_POSIX2_VERSION=199209`' in your environment.

3 Output of entire files

These commands read and write entire files, possibly transforming them in some way.

3.1 `cat`: Concatenate and write files

`cat` copies each *file* (‘`-`’ means standard input), or standard input if none are given, to standard output. Synopsis:

```
cat [option] [file]...
```

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

‘`-A`’
‘`--show-all`’
: Equivalent to ‘`-vET`’.

‘`-B`’
‘`--binary`’
: On MS-DOS and MS-Windows only, read and write the files in binary mode. By default, `cat` on MS-DOS/MS-Windows uses binary mode only when standard output is redirected to a file or a pipe; this option overrides that. Binary file I/O is used so that the files retain their format (Unix text as opposed to DOS text and binary), because `cat` is frequently used as a file-copying program. Some options (see below) cause `cat` to read and write files in text mode because in those cases the original file contents aren't important (e.g., when lines are numbered by `cat`, or when line endings should be marked). This is so these options work as DOS/Windows users would expect; for example, DOS-style text files have their lines end with the CR-LF pair of characters, which won't be processed as an empty line by ‘`-b`’ unless the file is read in text mode.

‘`-b`’
‘`--number-nonblank`’
: Number all nonblank output lines, starting with 1. On MS-DOS and MS-Windows, this option causes `cat` to read and write files in text mode.

‘`-e`’
: Equivalent to ‘`-vE`’.

‘`-E`’
‘`--show-ends`’
: Display a ‘`$`’ after the end of each line. On MS-DOS and MS-Windows, this option causes `cat` to read and write files in text mode.

‘`-n`’
‘`--number`’
: Number all output lines, starting with 1. On MS-DOS and MS-Windows, this option causes `cat` to read and write files in text mode.

‘`-s`’
‘`--squeeze-blank`’
: Replace multiple adjacent blank lines with a single blank line. On MS-DOS and MS-Windows, this option causes `cat` to read and write files in text mode.

'-t' Equivalent to '-vT'.

'-T'
'--show-tabs'
Display TAB characters as '^I'.

'-u' Ignored; for Unix compatibility.

'-v'
'--show-nonprinting'
Display control characters except for LFD and TAB using '^' notation and precede characters that have the high bit set with 'M-'. On MS-DOS and MS-Windows, this option causes `cat` to read files and standard input in DOS binary mode, so the CR characters at the end of each line are also visible.

3.2 `tac`: Concatenate and write files in reverse

`tac` copies each *file* ('-' means standard input), or standard input if none are given, to standard output, reversing the records (lines by default) in each separately. Synopsis:

```
tac [option]... [file]...
```

Records are separated by instances of a string (newline by default). By default, this separator string is attached to the end of the record that it follows in the file.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-b'
'--before'
The separator is attached to the beginning of the record that it precedes in the file.

'-r'
'--regex' Treat the separator string as a regular expression. Users of `tac` on MS-DOS/MS-Windows should note that, since `tac` reads files in binary mode, each line of a text file might end with a CR/LF pair instead of the Unix-style LF.

'-s *separator*'
'--separator=*separator*'
Use *separator* as the record separator, instead of newline.

3.3 `nl`: Number lines and write files

`nl` writes each *file* ('-' means standard input), or standard input if none are given, to standard output, with line numbers added to some or all of the lines. Synopsis:

```
nl [option]... [file]...
```

`nl` decomposes its input into (logical) pages; by default, the line number is reset to 1 at the top of each logical page. `nl` treats all of the input files as a single document; it does not reset line numbers or logical pages between files.

A logical page consists of three sections: header, body, and footer. Any of the sections can be empty. Each can be numbered in a different style from the others.

The beginnings of the sections of logical pages are indicated in the input file by a line containing exactly one of these delimiter strings:

'`\:\:\:`' start of header;

'`\:\:`' start of body;

'`\:`' start of footer.

The two characters from which these strings are made can be changed from '`\`' and '`:`' via options (see below), but the pattern and length of each string cannot be changed.

A section delimiter is replaced by an empty line on output. Any text that comes before the first section delimiter string in the input file is considered to be part of a body section, so `nl` treats a file that contains no section delimiters as a single body section.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-b` *style*'
'`--body-numbering=`*style*'
: Select the numbering style for lines in the body section of each logical page. When a line is not numbered, the current line number is not incremented, but the line number separator character is still prepended to the line. The styles are:

  '`a`' number all lines,

  '`t`' number only nonempty lines (default for body),

  '`n`' do not number lines (default for header and footer),

  '`p`*regexp*' number only lines that contain a match for *regexp*.

'`-d` *cd*'
'`--section-delimiter=`*cd*'
: Set the section delimiter characters to *cd*; default is '`\:`'. If only *c* is given, the second remains '`:`'. (Remember to protect '`\`' or other metacharacters from shell expansion with quotes or extra backslashes.)

'`-f` *style*'
'`--footer-numbering=`*style*'
: Analogous to '`--body-numbering`'.

'`-h` *style*'
'`--header-numbering=`*style*'
: Analogous to '`--body-numbering`'.

'`-i` *number*'
'`--page-increment=`*number*'
: Increment line numbers by *number* (default 1).

'-l *number*'
'--join-blank-lines=*number*'
: Consider *number* (default 1) consecutive empty lines to be one logical line for numbering, and only number the last one. Where fewer than *number* consecutive empty lines occur, do not number them. An empty line is one that contains no characters, not even spaces or tabs.

'-n *format*'
'--number-format=*format*'
: Select the line numbering format (default is `rn`):

  'ln' left justified, no leading zeros;

  'rn' right justified, no leading zeros;

  'rz' right justified, leading zeros.

'-p'
'--no-renumber'
: Do not reset the line number at the start of a logical page.

'-s *string*'
'--number-separator=*string*'
: Separate the line number from the text line in the output with *string* (default is the TAB character).

'-v *number*'
'--starting-line-number=*number*'
: Set the initial line number on each logical page to *number* (default 1).

'-w *number*'
'--number-width=*number*'
: Use *number* characters for line numbers (default 6).

3.4 `od`: Write files in octal or other formats

`od` writes an unambiguous representation of each *file* ('-' means standard input), or standard input if none are given. Synopses:

```
od [option]... [file]...
od --traditional [file] [[+]offset [[+]label]]
```

Each line of output consists of the offset in the input, followed by groups of data from the file. By default, `od` prints the offset in octal, and each group of file data is two bytes of input printed as a single octal number.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-A *radix*'
'--address-radix=*radix*'
: Select the base in which file offsets are printed. *radix* can be one of the following:

  'd' decimal;

'o' octal;

'x' hexadecimal;

'n' none (do not print offsets).

The default is octal.

'-j *bytes*'
'--skip-bytes=*bytes*'
: Skip *bytes* input bytes before formatting and writing. If *bytes* begins with '0x' or '0X', it is interpreted in hexadecimal; otherwise, if it begins with '0', in octal; otherwise, in decimal. Appending 'b' multiplies *bytes* by 512, 'k' by 1024, and 'm' by 1048576.

'-N *bytes*'
'--read-bytes=*bytes*'
: Output at most *bytes* bytes of the input. Prefixes and suffixes on `bytes` are interpreted as for the '-j' option.

'-s *n*'
'--strings[=*n*]'
: Instead of the normal output, output only *string constants*: at least *n* consecutive ASCII graphic characters, followed by a null (zero) byte.

If *n* is omitted with '--strings', the default is 3. On older systems, GNU `od` instead supports an obsolete option '-s[*n*]', where *n* also defaults to 3. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow '-s' without an argument; use '--strings' instead.

'-t *type*'
'--format=*type*'
: Select the format in which to output the file data. *type* is a string of one or more of the below type indicator characters. If you include more than one type indicator character in a single *type* string, or use this option more than once, `od` writes one copy of each output line using each of the data types that you specified, in the order that you specified.

Adding a trailing "z" to any type specification appends a display of the ASCII character representation of the printable characters to the output line generated by the type specification.

'a' named character

'c' ASCII character or backslash escape,

'd' signed decimal

'f' floating point

'o' octal

'u' unsigned decimal

'x' hexadecimal

The type `a` outputs things like '`sp`' for space, '`nl`' for newline, and '`nul`' for a null (zero) byte. Type `c` outputs ' ', '`\n`', and `\0`, respectively.

Except for types '`a`' and '`c`', you can specify the number of bytes to use in interpreting each number in the given data type by following the type indicator character with a decimal integer. Alternately, you can specify the size of one of the C compiler's built-in data types by following the type indicator character with one of the following characters. For integers ('`d`', '`o`', '`u`', '`x`'):

'`C`' char

'`S`' short

'`I`' int

'`L`' long

For floating point (`f`):

F float

D double

L long double

'`-v`'
'`--output-duplicates`'
Output consecutive lines that are identical. By default, when two or more consecutive output lines would be identical, `od` outputs only the first line, and puts just an asterisk on the following line to indicate the elision.

'`-w` *n*'
'`--width[=`*n*`]`'
Dump `n` input bytes per output line. This must be a multiple of the least common multiple of the sizes associated with the specified output types.

If this option is not given at all, the default is 16. If *n* is omitted with '`--width`', the default is 32. On older systems, GNU `od` instead supports an obsolete option '`-w[`*n*`]`', where *n* also defaults to 32. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow '`-w`' without an argument; use '`--width`' instead.

The next several options are shorthands for format specifications. GNU `od` accepts any combination of shorthands and format specification options. These options accumulate.

'`-a`' Output as named characters. Equivalent to '`-ta`'.

'`-b`' Output as octal bytes. Equivalent to '`-toC`'.

'`-c`' Output as ASCII characters or backslash escapes. Equivalent to '`-tc`'.

'`-d`' Output as unsigned decimal shorts. Equivalent to '`-tu2`'.

'`-f`' Output as floats. Equivalent to '`-tfF`'.

'`-h`' Output as hexadecimal shorts. Equivalent to '`-tx2`'.

'`-i`' Output as decimal shorts. Equivalent to '`-td2`'.

'-l' Output as decimal longs. Equivalent to '-td4'.

'-o' Output as octal shorts. Equivalent to '-to2'.

'-x' Output as hexadecimal shorts. Equivalent to '-tx2'.

'--traditional'
Recognize the non-option arguments that traditional od accepted. The following syntax:

od --traditional [*file*] [[+]*offset*[.][b] [[+]*label*[.][b]]]

can be used to specify at most one file and optional arguments specifying an offset and a pseudo-start address, *label*. By default, *offset* is interpreted as an octal number specifying how many input bytes to skip before formatting and writing. The optional trailing decimal point forces the interpretation of *offset* as a decimal number. If no decimal is specified and the offset begins with '0x' or '0X' it is interpreted as a hexadecimal number. If there is a trailing 'b', the number of bytes skipped will be *offset* multiplied by 512. The *label* argument is interpreted just like *offset*, but it specifies an initial pseudo-address. The pseudo-addresses are displayed in parentheses following any normal address.

4 Formatting file contents

These commands reformat the contents of files.

4.1 `fmt`: Reformat paragraph text

`fmt` fills and joins lines to produce output lines of (at most) a given number of characters (75 by default). Synopsis:

```
fmt [option]... [file]...
```

`fmt` reads from the specified *file* arguments (or standard input if none are given), and writes to standard output.

By default, blank lines, spaces between words, and indentation are preserved in the output; successive input lines with different indentation are not joined; tabs are expanded on input and introduced on output.

`fmt` prefers breaking lines at the end of a sentence, and tries to avoid line breaks after the first word of a sentence or before the last word of a sentence. A *sentence break* is defined as either the end of a paragraph or a word ending in any of '`.?!`', followed by two spaces or end of line, ignoring any intervening parentheses or quotes. Like TEX, `fmt` reads entire "paragraphs" before choosing line breaks; the algorithm is a variant of that in "Breaking Paragraphs Into Lines" (Donald E. Knuth and Michael F. Plass, *Software—Practice and Experience*, 11 (1981), 1119–1184).

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-c`'
'`--crown-margin`'
: *Crown margin* mode: preserve the indentation of the first two lines within a paragraph, and align the left margin of each subsequent line with that of the second line.

'`-t`'
'`--tagged-paragraph`'
: *Tagged paragraph* mode: like crown margin mode, except that if indentation of the first line of a paragraph is the same as the indentation of the second, the first line is treated as a one-line paragraph.

'`-s`'
'`--split-only`'
: Split lines only. Do not join short lines to form longer ones. This prevents sample lines of code, and other such "formatted" text from being unduly combined.

'`-u`'
'`--uniform-spacing`'
: Uniform spacing. Reduce spacing between words to one space, and spacing between sentences to two spaces.

'-*width*'
'-w *width*'
'--width=*width*'
: Fill output lines up to *width* characters (default 75). `fmt` initially tries to make lines about 7% shorter than this, to give it room to balance line lengths.

'-p *prefix*'
'--prefix=*prefix*'
: Only lines beginning with *prefix* (possibly preceded by whitespace) are subject to formatting. The prefix and any preceding whitespace are stripped for the formatting and then re-attached to each formatted output line. One use is to format certain kinds of program comments, while leaving the code unchanged.

4.2 pr: Paginate or columnate files for printing

`pr` writes each *file* ('-' means standard input), or standard input if none are given, to standard output, paginating and optionally outputting in multicolumn format; optionally merges all *file*s, printing all in parallel, one per column. Synopsis:

```
pr [option]... [file]...
```

By default, a 5-line header is printed at each page: two blank lines; a line with the date, the filename, and the page count; and two more blank lines. A footer of five blank lines is also printed. With the '-F' option, a 3-line header is printed: the leading two blank lines are omitted; no footer is used. The default *page_length* in both cases is 66 lines. The default number of text lines changes from 56 (without '-F') to 63 (with '-F'). The text line of the header takes the form '*date string page*', with spaces inserted around *string* so that the line takes up the full *page_width*. Here, *date* is the date (see the '-D' or '--date-format' option for details), *string* is the centered header string, and *page* identifies the page number. The `LC_MESSAGES` locale category affects the spelling of *page*; in the default C locale, it is 'Page *number*' where *number* is the decimal page number.

Form feeds in the input cause page breaks in the output. Multiple form feeds produce empty pages.

Columns are of equal width, separated by an optional string (default is 'space'). For multicolumn output, lines will always be truncated to *page_width* (default 72), unless you use the '-J' option. For single column output no line truncation occurs by default. Use '-W' option to truncate lines in that case.

The following changes were made in version 1.22i and apply to later versions of `pr`: - Brian

- Some small *letter options* ('-s', '-w') have been redefined for better POSIX compliance. The output of some further cases has been adapted to other Unix systems. These changes are not compatible with earlier versions of the program.
- Some *new capital letter* options ('-J', '-S', '-W') have been introduced to turn off unexpected interferences of small letter options. The '-N' option and the second argument *last_page* of '+FIRST_PAGE' offer more flexibility. The detailed handling of form feeds set in the input files requires the '-T' option.
- Capital letter options override small letter ones.

- Some of the option-arguments (compare '`-s`', '`-e`', '`-i`', '`-n`') cannot be specified as separate arguments from the preceding option letter (already stated in the POSIX specification).

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`+first_page[:last_page]`'
'`--pages=first_page[:last_page]`'
: Begin printing with page *first_page* and stop with *last_page*. Missing '`:last_page`' implies end of file. While estimating the number of skipped pages each form feed in the input file results in a new page. Page counting with and without '`+first_page`' is identical. By default, counting starts with the first page of input file (not first page printed). Line numbering may be altered by '`-N`' option.

'`-column`'
'`--columns=column`'
: With each single *file*, produce *column* columns of output (default is 1) and print columns down, unless '`-a`' is used. The column width is automatically decreased as *column* increases; unless you use the '`-W/-w`' option to increase *page_width* as well. This option might well cause some lines to be truncated. The number of lines in the columns on each page are balanced. The options '`-e`' and '`-i`' are on for multiple text-column output. Together with '`-J`' option column alignment and line truncation is turned off. Lines of full length are joined in a free field format and '`-S`' option may set field separators. '`-column`' may not be used with '`-m`' option.

'`-a`'
'`--across`'
: With each single *file*, print columns across rather than down. The '`-column`' option must be given with *column* greater than one. If a line is too long to fit in a column, it is truncated.

'`-c`'
'`--show-control-chars`'
: Print control characters using hat notation (e.g., '`^G`'); print other nonprinting characters in octal backslash notation. By default, nonprinting characters are not changed.

'`-d`'
'`--double-space`'
: Double space the output.

'`-D format`'
'`--date-format=format`'
: Format header dates using *format*, using the same conventions as for the the command '`date +format`'; See Section 21.1 [date invocation], page 117. Except for directives, which start with '`%`', characters in *format* are printed unchanged. You can use this option to specify an arbitrary string in place of the header date, e.g., '`--date-format="Monday morning"`'.

If the POSIXLY_CORRECT environment variable is not set, the date format defaults to '%Y-%m-%d %H:%M' (for example, '2001-12-04 23:59'); otherwise, the format depends on the LC_TIME locale category, with the default being '%b %e %H:%M %Y' (for example, 'Dec 4 23:59 2001'.

'-e[*in-tabchar*[*in-tabwidth*]]'
'--expand-tabs[=*in-tabchar*[*in-tabwidth*]]'
: Expand *tab*s to spaces on input. Optional argument *in-tabchar* is the input tab character (default is the TAB character). Second optional argument *in-tabwidth* is the input tab character's width (default is 8).

'-f'
'-F'
'--form-feed'
: Use a form feed instead of newlines to separate output pages. The default page length of 66 lines is not altered. But the number of lines of text per page changes from default 56 to 63 lines.

'-h *HEADER*'
'--header=*HEADER*'
: Replace the filename in the header with the centered string *header*. When using the shell, *header* should be quoted and should be separated from '-h' by a space.

'-i[*out-tabchar*[*out-tabwidth*]]'
'--output-tabs[=*out-tabchar*[*out-tabwidth*]]'
: Replace spaces with *tab*s on output. Optional argument *out-tabchar* is the output tab character (default is the TAB character). Second optional argument *out-tabwidth* is the output tab character's width (default is 8).

'-J'
'--join-lines'
: Merge lines of full length. Used together with the column options '-*column*', '-a -*column*' or '-m'. Turns off '-W/-w' line truncation; no column alignment used; may be used with '--sep-string[=*string*]'. '-J' has been introduced (together with '-W' and '--sep-string') to disentangle the old (POSIX-compliant) options '-w' and '-s' along with the three column options.

'-l *page_length*'
'--length=*page_length*'
: Set the page length to *page_length* (default 66) lines, including the lines of the header [and the footer]. If *page_length* is less than or equal to 10 (or <= 3 with '-F'), the header and footer are omitted, and all form feeds set in input files are eliminated, as if the '-T' option had been given.

'-m'
'--merge'
: Merge and print all *file*s in parallel, one in each column. If a line is too long to fit in a column, it is truncated, unless the '-J' option is used. '--sep-string[=*string*]' may be used. Empty pages in some *file*s (form feeds set) produce empty columns, still marked by *string*. The result is a continuous line numbering and column marking throughout the whole merged

file. Completely empty merged pages show no separators or line numbers. The default header becomes '*date page*' with spaces inserted in the middle; this may be used with the '-h' or '--header' option to fill up the middle blank part.

'-n[*number-separator*[*digits*]]'
'--number-lines[=*number-separator*[*digits*]]'

Provide *digits* digit line numbering (default for *digits* is 5). With multicolumn output the number occupies the first *digits* column positions of each text column or only each line of '-m' output. With single column output the number precedes each line just as '-m' does. Default counting of the line numbers starts with the first line of the input file (not the first line printed, compare the '--page' option and '-N' option). Optional argument *number-separator* is the character appended to the line number to separate it from the text followed. The default separator is the TAB character. In a strict sense a TAB is always printed with single column output only. The *TAB*-width varies with the *TAB*-position, e.g. with the left *margin* specified by '-o' option. With multicolumn output priority is given to 'equal width of output columns' (a POSIX specification). The *TAB*-width is fixed to the value of the first column and does not change with different values of left *margin*. That means a fixed number of spaces is always printed in the place of the *number-separator tab*. The tabification depends upon the output position.

'-N *line_number*'
'--first-line-number=*line_number*'

Start line counting with the number *line_number* at first line of first page printed (in most cases not the first line of the input file).

'-o *margin*'
'--indent=*margin*'

Indent each line with a margin *margin* spaces wide (default is zero). The total page width is the size of the margin plus the *page_width* set with the '-W/-w' option. A limited overflow may occur with numbered single column output (compare '-n' option).

'-r'
'--no-file-warnings'

Do not print a warning message when an argument *file* cannot be opened. (The exit status will still be nonzero, however.)

'-s[*char*]'
'--separator[=*char*]'

Separate columns by a single character *char*. The default for *char* is the TAB character without '-w' and 'no character' with '-w'. Without '-s' the default separator 'space' is set. '-s[char]' turns off line truncation of all three column options ('-COLUMN'|'-a -COLUMN'|'-m') unless '-w' is set. This is a POSIX-compliant formulation.

'-S *string*'
'--sep-string[=*string*]'
Use *string* to separate output columns. The '-S' option doesn't affect the '-W/-w' option, unlike the '-s' option which does. It does not affect line truncation or column alignment. Without '-S', and with '-J', pr uses the default output separator, TAB. Without '-S' or '-J', pr uses a 'space' (same as '-S" "'). With '-S*string*', *string* must be nonempty; '--sep-string' with no *string* is equivalent to '--sep-string=""'.

On older systems, pr instead supports an obsolete option '-S[*string*]', where *string* is optional. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this older usage. To specify an empty *string* portably, use '--sep-string'.

'-t'
'--omit-header'
Do not print the usual header [and footer] on each page, and do not fill out the bottom of pages (with blank lines or a form feed). No page structure is produced, but form feeds set in the input files are retained. The predefined pagination is not changed. '-t' or '-T' may be useful together with other options; e.g.: '-t -e4', expand TAB characters in the input file to 4 spaces but don't make any other changes. Use of '-t' overrides '-h'.

'-T'
'--omit-pagination'
Do not print header [and footer]. In addition eliminate all form feeds set in the input files.

'-v'
'--show-nonprinting'
Print nonprinting characters in octal backslash notation.

'-w *page_width*'
'--width=*page_width*'
Set page width to *page_width* characters for multiple text-column output only (default for *page_width* is 72). '-s[CHAR]' turns off the default page width and any line truncation and column alignment. Lines of full length are merged, regardless of the column options set. No *page_width* setting is possible with single column output. A POSIX-compliant formulation.

'-W *page_width*'
'--page_width=*page_width*'
Set the page width to *page_width* characters. That's valid with and without a column option. Text lines are truncated, unless '-J' is used. Together with one of the three column options ('-*column*', '-a -*column*' or '-m') column alignment is always used. The separator options '-S' or '-s' don't affect the '-W' option. Default is 72 characters. Without '-W *page_width*' and without any of the column options NO line truncation is used (defined to keep downward compatibility and to meet most frequent tasks). That's equivalent to '-W 72 -J'. The header line is never truncated.

4.3 `fold`: Wrap input lines to fit in specified width

`fold` writes each *file* ('`-`' means standard input), or standard input if none are given, to standard output, breaking long lines. Synopsis:

```
fold [option]... [file]...
```

By default, `fold` breaks lines wider than 80 columns. The output is split into as many lines as necessary.

`fold` counts screen columns by default; thus, a tab may count more than one column, backspace decreases the column count, and carriage return sets the column to zero.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-b`'
'`--bytes`'
: Count bytes rather than columns, so that tabs, backspaces, and carriage returns are each counted as taking up one column, just like other characters.

'`-s`'
'`--spaces`'
: Break at word boundaries: the line is broken after the last blank before the maximum line length. If the line contains no such blanks, the line is broken at the maximum line length as usual.

'`-w` ***width***'
'`--width=`***width***'
: Use a maximum line length of *width* columns instead of 80.

: On older systems, `fold` supports an obsolete option '`-`***width***'. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '`-w` ***width***' instead.

5 Output of parts of files

These commands output pieces of the input.

5.1 `head`: Output the first part of files

`head` prints the first part (10 lines by default) of each *file*; it reads from standard input if no files are given or when given a *file* of '`-`'. Synopsis:

```
head [option]... [file]...
```

If more than one *file* is specified, `head` prints a one-line header consisting of

```
==> file name <==
```

before the output for each *file*.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-c bytes`'
'`--bytes=bytes`'
: Print the first *bytes* bytes, instead of initial lines. Appending '`b`' multiplies *bytes* by 512, '`k`' by 1024, and '`m`' by 1048576.

'`-n n`'
'`--lines=n`'
: Output the first *n* lines.

'`-q`'
'`--quiet`'
'`--silent`'
: Never print file name headers.

'`-v`'
'`--verbose`'
: Always print file name headers.

On older systems, `head` supports an obsolete option '`-countoptions`', which is recognized only if it is specified first. *count* is a decimal number optionally followed by a size letter ('`b`', '`k`', '`m`') as in `-c`, or '`l`' to mean count by lines, or other option letters ('`cqv`'). POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '`-c count`' or '`-n count`' instead.

5.2 `tail`: Output the last part of files

`tail` prints the last part (10 lines by default) of each *file*; it reads from standard input if no files are given or when given a *file* of '`-`'. Synopsis:

```
tail [option]... [file]...
```

If more than one *file* is specified, `tail` prints a one-line header consisting of

```
==> file name <==
```

before the output for each *file*.

GNU `tail` can output any amount of data (some other versions of `tail` cannot). It also has no '`-r`' option (print in reverse), since reversing a file is really a different job from printing the end of a file; BSD `tail` (which is the one with `-r`) can only reverse files that are at most as large as its buffer, which is typically 32 KiB. A more reliable and versatile way to reverse files is the GNU `tac` command.

If any option-argument is a number *n* starting with a '`+`', `tail` begins printing with the *n*th item from the start of each file, instead of from the end.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-c` *bytes*'
'`--bytes=`*bytes*'
: Output the last *bytes* bytes, instead of final lines. Appending '`b`' multiplies *bytes* by 512, '`k`' by 1024, and '`m`' by 1048576.

'`-f`'
'`--follow[=`*how*`]`'
: Loop forever trying to read more characters at the end of the file, presumably because the file is growing. This option is ignored when reading from a pipe. If more than one file is given, `tail` prints a header whenever it gets output from a different file, to indicate which file that output is from.

There are two ways to specify how you'd like to track files with this option, but that difference is noticeable only when a followed file is removed or renamed. If you'd like to continue to track the end of a growing file even after it has been unlinked, use '`--follow=descriptor`'. This is the default behavior, but it is not useful if you're tracking a log file that may be rotated (removed or renamed, then reopened). In that case, use '`--follow=name`' to track the named file by reopening it periodically to see if it has been removed and recreated by some other program.

No matter which method you use, if the tracked file is determined to have shrunk, `tail` prints a message saying the file has been truncated and resumes tracking the end of the file from the newly-determined endpoint.

When a file is removed, `tail`'s behavior depends on whether it is following the name or the descriptor. When following by name, tail can detect that a file has been removed and gives a message to that effect, and if '`--retry`' has been specified it will continue checking periodically to see if the file reappears. When following a descriptor, tail does not detect that the file has been unlinked or renamed and issues no message; even though the file may no longer be accessible via its original name, it may still be growing.

The option values '`descriptor`' and '`name`' may be specified only with the long form of the option, not with '`-f`'.

'-F' This option is the same as '--follow=name --retry'. That is, tail will attempt to reopen a file when it is removed. Should this fail, tail will keep trying until it becomes accessible again.

'--retry' This option is meaningful only when following by name. Without this option, when tail encounters a file that doesn't exist or is otherwise inaccessible, it reports that fact and never checks it again.

'--sleep-interval=*number*'
: Change the number of seconds to wait between iterations (the default is 1.0). During one iteration, every specified file is checked to see if it has Historical implementations of `tail` have required that *number* be an integer. However, GNU `tail` accepts an arbitrary floating point number.

'--pid=*pid*'
: When following by name or by descriptor, you may specify the process ID, *pid*, of the sole writer of all *file* arguments. Then, shortly after that process terminates, tail will also terminate. This will work properly only if the writer and the tailing process are running on the same machine. For example, to save the output of a build in a file and to watch the file grow, if you invoke `make` and `tail` like this then the tail process will stop when your build completes. Without this option, you would have had to kill the `tail -f` process yourself.

```
$ make >& makerr & tail --pid=$! -f makerr
```

If you specify a *pid* that is not in use or that does not correspond to the process that is writing to the tailed files, then `tail` may terminate long before any *file*s stop growing or it may not terminate until long after the real writer has terminated. Note that '--pid' cannot be supported on some systems; `tail` will print a warning if this is the case.

'--max-unchanged-stats=*n*'
: When tailing a file by name, if there have been *n* (default n=5) consecutive iterations for which the size has remained the same, then `open`/`fstat` the file to determine if that file name is still associated with the same device/inode-number pair as before. When following a log file that is rotated, this is approximately the number of seconds between when tail prints the last pre-rotation lines and when it prints the lines that have accumulated in the new log file. This option is meaningful only when following by name.

'-n *n*'
'--lines=*n*'
: Output the last *n* lines.

'-q'
'--quiet'
'--silent'
: Never print file name headers.

'-v'
'--verbose'
: Always print file name headers.

On older systems, `tail` supports an obsolete option '-*count* *options*', which is recognized only if it is specified first. *count* is a decimal number optionally followed by a size letter ('`b`', '`k`', '`m`') as in `-c`, or '`l`' to mean count by lines, or other option letters ('`cfqv`'). Some older `tail` implementations also support an obsolete option '`+count`' with the same meaning as '`-+count`'. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow these options; use '`-c count`' or '`-n count`' instead.

5.3 `split`: Split a file into fixed-size pieces

`split` creates output files containing consecutive sections of *input* (standard input if none is given or *input* is '`-`'). Synopsis:

```
split [option] [input [prefix]]
```

By default, `split` puts 1000 lines of *input* (or whatever is left over for the last section), into each output file.

The output files' names consist of *prefix* ('`x`' by default) followed by a group of letters ('`aa`', '`ab`', . . . by default), such that concatenating the output files in sorted order by file name produces the original input file. If the output file names are exhausted, `split` reports an error without deleting the output files that it did create.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-a length`'
'`--suffix-length=length`'
: Use suffixes of length *length*. The default *length* is 2.

'`-l lines`'
'`--lines=lines`'
: Put *lines* lines of *input* into each output file.

  On older systems, `split` supports an obsolete option '`-lines`'. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '`-l lines`' instead.

'`-b bytes`'
'`--bytes=bytes`'
: Put the first *bytes* bytes of *input* into each output file. Appending '`b`' multiplies *bytes* by 512, '`k`' by 1024, and '`m`' by 1048576.

'`-C bytes`'
'`--line-bytes=bytes`'
: Put into each output file as many complete lines of *input* as possible without exceeding *bytes* bytes. For lines longer than *bytes* bytes, put *bytes* bytes into each output file until less than *bytes* bytes of the line are left, then continue normally. *bytes* has the same format as for the '`--bytes`' option.

'`--verbose`'
: Write a diagnostic to standard error just before each output file is opened.

5.4 `csplit`: Split a file into context-determined pieces

`csplit` creates zero or more output files containing sections of *input* (standard input if *input* is '`-`'). Synopsis:

```
csplit [option]... input pattern...
```

The contents of the output files are determined by the *pattern* arguments, as detailed below. An error occurs if a *pattern* argument refers to a nonexistent line of the input file (e.g., if no remaining line matches a given regular expression). After every *pattern* has been matched, any remaining input is copied into one last output file.

By default, `csplit` prints the number of bytes written to each output file after it has been created.

The types of pattern arguments are:

'*n*'
: Create an output file containing the input up to but not including line *n* (a positive integer). If followed by a repeat count, also create an output file containing the next *line* lines of the input file once for each repeat.

'`/`*regexp*`/[`*offset*`]`'
: Create an output file containing the current line up to (but not including) the next line of the input file that contains a match for *regexp*. The optional *offset* is a '`+`' or '`-`' followed by a positive integer. If it is given, the input up to the matching line plus or minus *offset* is put into the output file, and the line after that begins the next section of input.

'`%`*regexp*`%[`*offset*`]`'
: Like the previous type, except that it does not create an output file, so that section of the input file is effectively ignored.

'`{`*repeat-count*`}`'
: Repeat the previous pattern *repeat-count* additional times. *repeat-count* can either be a positive integer or an asterisk, meaning repeat as many times as necessary until the input is exhausted.

The output files' names consist of a prefix ('`xx`' by default) followed by a suffix. By default, the suffix is an ascending sequence of two-digit decimal numbers from '`00`' to '`99`'. In any case, concatenating the output files in sorted order by filename produces the original input file.

By default, if `csplit` encounters an error or receives a hangup, interrupt, quit, or terminate signal, it removes any output files that it has created so far before it exits.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-f` *prefix*'
'`--prefix=`*prefix*'
: Use *prefix* as the output file name prefix.

'`-b` *suffix*'
'`--suffix=`*suffix*'
: Use *suffix* as the output file name suffix. When this option is specified, the suffix string must include exactly one `printf(3)`-style conversion specification,

possibly including format specification flags, a field width, a precision specifications, or all of these kinds of modifiers. The format letter must convert a binary integer argument to readable form; thus, only '`d`', '`i`', '`u`', '`o`', '`x`', and '`X`' conversions are allowed. The entire *suffix* is given (with the current output file number) to `sprintf(3)` to form the file name suffixes for each of the individual output files in turn. If this option is used, the '`--digits`' option is ignored.

'`-n` *digits*'
'`--digits=`*digits*'
Use output file names containing numbers that are *digits* digits long instead of the default 2.

'`-k`'
'`--keep-files`'
Do not remove output files when errors are encountered.

'`-z`'
'`--elide-empty-files`'
Suppress the generation of zero-length output files. (In cases where the section delimiters of the input file are supposed to mark the first lines of each of the sections, the first output file will generally be a zero-length file unless you use this option.) The output file sequence numbers always run consecutively starting from 0, even when this option is specified.

'`-s`'
'`-q`'
'`--silent`'
'`--quiet`' Do not print counts of output file sizes.

6 Summarizing files

These commands generate just a few numbers representing entire contents of files.

6.1 wc: Print byte, word, and line counts

`wc` counts the number of bytes, characters, whitespace-separated words, and newlines in each given *file*, or standard input if none are given or for a *file* of '`-`'. Synopsis:

```
wc [option]... [file]...
```

`wc` prints one line of counts for each file, and if the file was given as an argument, it prints the file name following the counts. If more than one *file* is given, `wc` prints a final line containing the cumulative counts, with the file name '`total`'. The counts are printed in this order: newlines, words, characters, bytes. By default, each count is output right-justified in a 7-byte field with one space between fields so that the numbers and file names line up nicely in columns. However, POSIX requires that there be exactly one space separating columns. You can make `wc` use the POSIX-mandated output format by setting the `POSIXLY_CORRECT` environment variable.

By default, `wc` prints three counts: the newline, words, and byte counts. Options can specify that only certain counts be printed. Options do not undo others previously given, so

```
wc --bytes --words
```

prints both the byte counts and the word counts.

With the `--max-line-length` option, `wc` prints the length of the longest line per file, and if there is more than one file it prints the maximum (not the sum) of those lengths.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-c`'
'`--bytes`' Print only the byte counts.

'`-m`'
'`--chars`' Print only the character counts.

'`-w`'
'`--words`' Print only the word counts.

'`-l`'
'`--lines`' Print only the newline counts.

'`-L`'
'`--max-line-length`'
Print only the maximum line lengths.

6.2 `sum`: Print checksum and block counts

`sum` computes a 16-bit checksum for each given *file*, or standard input if none are given or for a *file* of '`-`'. Synopsis:

```
sum [option]... [file]...
```

`sum` prints the checksum for each *file* followed by the number of blocks in the file (rounded up). If more than one *file* is given, file names are also printed (by default). (With the '`--sysv`' option, corresponding file names are printed when there is at least one file argument.)

By default, GNU `sum` computes checksums using an algorithm compatible with BSD `sum` and prints file sizes in units of 1024-byte blocks.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-r`' Use the default (BSD compatible) algorithm. This option is included for compatibility with the System V `sum`. Unless '`-s`' was also given, it has no effect.

'`-s`'
'`--sysv`' Compute checksums using an algorithm compatible with System V `sum`'s default, and print file sizes in units of 512-byte blocks.

`sum` is provided for compatibility; the `cksum` program (see next section) is preferable in new applications.

6.3 `cksum`: Print CRC checksum and byte counts

`cksum` computes a cyclic redundancy check (CRC) checksum for each given *file*, or standard input if none are given or for a *file* of '`-`'. Synopsis:

```
cksum [option]... [file]...
```

`cksum` prints the CRC checksum for each file along with the number of bytes in the file, and the filename unless no arguments were given.

`cksum` is typically used to ensure that files transferred by unreliable means (e.g., netnews) have not been corrupted, by comparing the `cksum` output for the received files with the `cksum` output for the original files (typically given in the distribution).

The CRC algorithm is specified by the POSIX standard. It is not compatible with the BSD or System V `sum` algorithms (see the previous section); it is more robust.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

6.4 `md5sum`: Print or check message-digests

`md5sum` computes a 128-bit checksum (or *fingerprint* or *message-digest*) for each specified *file*. If a *file* is specified as '`-`' or if no files are given `md5sum` computes the checksum for the standard input. `md5sum` can also determine whether a file and checksum are consistent. Synopses:

```
md5sum [option]... [file]...
md5sum [option]... --check [file]
```

For each *file*, '`md5sum`' outputs the MD5 checksum, a flag indicating a binary or text input file, and the filename. If *file* is omitted or specified as '`-`', standard input is read.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-b`'
'`--binary`'
: Treat all input files as binary. This option has no effect on Unix systems, since they don't distinguish between binary and text files. This option is useful on systems that have different internal and external character representations. On MS-DOS and MS-Windows, this is the default.

'`-c`'
'`--check`'
: Read filenames and checksum information from the single *file* (or from stdin if no *file* was specified) and report whether each named file and the corresponding checksum data are consistent. The input to this mode of `md5sum` is usually the output of a prior, checksum-generating run of '`md5sum`'. Each valid line of input consists of an MD5 checksum, a binary/text flag, and then a filename. Binary files are marked with '`*`', text with '` `'. For each such line, `md5sum` reads the named file and computes its MD5 checksum. Then, if the computed message digest does not match the one on the line with the filename, the file is noted as having failed the test. Otherwise, the file passes the test. By default, for each valid line, one line is written to standard output indicating whether the named file passed the test. After all checks have been performed, if there were any failures, a warning is issued to standard error. Use the '`--status`' option to inhibit that output. If any listed file cannot be opened or read, if any valid line has an MD5 checksum inconsistent with the associated file, or if no valid line is found, `md5sum` exits with nonzero status. Otherwise, it exits successfully.

'`--status`'
: This option is useful only when verifying checksums. When verifying checksums, don't generate the default one-line-per-file diagnostic and don't output the warning summarizing any failures. Failures to open or read a file still evoke individual diagnostics to standard error. If all listed files are readable and are consistent with the associated MD5 checksums, exit successfully. Otherwise exit with a status code indicating there was a failure.

'`-t`'
'`--text`'
: Treat all input files as text files. This is the reverse of '`--binary`'.

'`-w`'
'`--warn`'
: When verifying checksums, warn about improperly formatted MD5 checksum lines. This option is useful only if all but a few lines in the checked input are valid.

7 Operating on sorted files

These commands work with (or produce) sorted files.

7.1 sort: Sort text files

`sort` sorts, merges, or compares all the lines from the given files, or standard input if none are given or for a *file* of '-'. By default, `sort` writes the results to standard output. Synopsis:

```
sort [option]... [file]...
```

`sort` has three modes of operation: sort (the default), merge, and check for sortedness. The following options change the operation mode:

'-c'
'--check' Check whether the given files are already sorted: if they are not all sorted, print an error message and exit with a status of 1. Otherwise, exit successfully.

'-m'
'--merge' Merge the given files by sorting them as a group. Each input file must always be individually sorted. It always works to sort instead of merge; merging is provided because it is faster, in the case where it works.

A pair of lines is compared as follows: if any key fields have been specified, `sort` compares each pair of fields, in the order specified on the command line, according to the associated ordering options, until a difference is found or no fields are left. Unless otherwise specified, all comparisons use the character collating sequence specified by the `LC_COLLATE` locale.[1]

If any of the global options '`bdfgiMnr`' are given but no key fields are specified, `sort` compares the entire lines according to the global options.

Finally, as a last resort when all keys compare equal (or if no ordering options were specified at all), `sort` compares the entire lines. The last resort comparison honors the '`--reverse`' ('`-r`') global option. The '`--stable`' ('`-s`') option disables this last-resort comparison so that lines in which all fields compare equal are left in their original relative order. If no fields or global options are specified, '`--stable`' ('`-s`') has no effect.

GNU `sort` (as specified for all GNU utilities) has no limits on input line length or restrictions on bytes allowed within lines. In addition, if the final byte of an input file is not a newline, GNU `sort` silently supplies one. A line's trailing newline is not part of the line for comparison purposes.

Upon any error, `sort` exits with a status of '2'.

If the environment variable `TMPDIR` is set, `sort` uses its value as the directory for temporary files instead of '`/tmp`'. The '`--temporary-directory`' ('`-T`') option in turn overrides the environment variable.

---

[1] If you use a non-POSIX locale (e.g., by setting `LC_ALL` to '`en_US`'), then `sort` may produce output that is sorted differently than you're accustomed to. In that case, set the `LC_ALL` environment variable to '`C`'. Note that setting only `LC_COLLATE` has two problems. First, it is ineffective if `LC_ALL` is also set. Second, it has undefined behavior if `LC_CTYPE` (or `LANG`, if `LC_CTYPE` is unset) is set to an incompatible value. For example, you get undefined behavior if `LC_CTYPE` is `ja_JP.PCK` but `LC_COLLATE` is `en_US.UTF-8`.

The following options affect the ordering of output lines. They may be specified globally or as part of a specific key field. If no key fields are specified, global options apply to comparison of entire lines; otherwise the global options are inherited by key fields that do not specify any special options of their own. In pre-POSIX versions of `sort`, global options affect only later key fields, so portable shell scripts should specify global options first.

'`-b`'
'`--ignore-leading-blanks`'
: Ignore leading blanks when finding sort keys in each line. The `LC_CTYPE` locale determines character types.

'`-d`'
'`--dictionary-order`'
: Sort in *phone directory* order: ignore all characters except letters, digits and blanks when sorting. The `LC_CTYPE` locale determines character types.

'`-f`'
'`--ignore-case`'
: Fold lowercase characters into the equivalent uppercase characters when comparing so that, for example, '`b`' and '`B`' sort as equal. The `LC_CTYPE` locale determines character types.

'`-g`'
'`--general-numeric-sort`'
: Sort numerically, using the standard C function `strtod` to convert a prefix of each line to a double-precision floating point number. This allows floating point numbers to be specified in scientific notation, like `1.0e-34` and `10e100`. The `LC_NUMERIC` locale determines the decimal-point character. Do not report overflow, underflow, or conversion errors. Use the following collating sequence:

- Lines that do not start with numbers (all considered to be equal).
- NaNs ("Not a Number" values, in IEEE floating point arithmetic) in a consistent but machine-dependent order.
- Minus infinity.
- Finite numbers in ascending numeric order (with -0 and $+0$ equal).
- Plus infinity.

Use this option only if there is no alternative; it is much slower than '`--numeric-sort`' ('`-n`') and it can lose information when converting to floating point.

'`-i`'
'`--ignore-nonprinting`'
: Ignore nonprinting characters. The `LC_CTYPE` locale determines character types.

'`-M`'
'`--month-sort`'
: An initial string, consisting of any amount of whitespace, followed by a month name abbreviation, is folded to UPPER case and compared in the order '`JAN`' < '`FEB`' < . . . < '`DEC`'. Invalid names compare low to valid names. The `LC_TIME` locale category determines the month spellings.

'-n'
'--numeric-sort'

Sort numerically: the number begins each line; specifically, it consists of optional whitespace, an optional '-' sign, and zero or more digits possibly separated by thousands separators, optionally followed by a decimal-point character and zero or more digits. The `LC_NUMERIC` locale specifies the decimal-point character and thousands separator.

Numeric sort uses what might be considered an unconventional method to compare strings representing floating point numbers. Rather than first converting each string to the C `double` type and then comparing those values, `sort` aligns the decimal-point characters in the two strings and compares the strings a character at a time. One benefit of using this approach is its speed. In practice this is much more efficient than performing the two corresponding string-to-double (or even string-to-integer) conversions and then comparing doubles. In addition, there is no corresponding loss of precision. Converting each string to `double` before comparison would limit precision to about 16 digits on most systems.

Neither a leading '+' nor exponential notation is recognized. To compare such strings numerically, use the '--general-numeric-sort' ('-g') option.

'-r'
'--reverse'

Reverse the result of comparison, so that lines with greater key values appear earlier in the output instead of later.

Other options are:

'-o *output-file*'
'--output=*output-file*'

Write output to *output-file* instead of standard output. If necessary, `sort` reads input before opening *output-file*, so you can safely sort a file in place by using commands like `sort -o F F` and `cat F | sort -o F`.

On newer systems, '-o' cannot appear after an input file if `POSIXLY_CORRECT` is set, e.g., 'sort F -o F'. Portable scripts should specify '-o *output-file*' before any input files.

'-S *size*'
'--buffer-size=*size*'

Use a main-memory sort buffer of the given *size*. By default, *size* is in units of 1024 bytes. Appending '%' causes *size* to be interpreted as a percentage of physical memory. Appending 'K' multiplies *size* by 1024 (the default), 'M' by 1,048,576, 'G' by 1,073,741,824, and so on for 'T', 'P', 'E', 'Z', and 'Y'. Appending 'b' causes *size* to be interpreted as a byte count, with no multiplication.

This option can improve the performance of `sort` by causing it to start with a larger or smaller sort buffer than the default. However, this option affects only the initial buffer size. The buffer grows beyond *size* if `sort` encounters input lines larger than *size*.

'`-t` *separator*'
'`--field-separator=`*separator*'

Use character *separator* as the field separator when finding the sort keys in each line. By default, fields are separated by the empty string between a non-whitespace character and a whitespace character. That is, given the input line '` foo bar`', `sort` breaks it into fields '` foo`' and '` bar`'. The field separator is not considered to be part of either the field preceding or the field following. But note that sort fields that extend to the end of the line, as '`-k 2`', or sort fields consisting of a range, as '`-k 2,3`', retain the field separators present between the endpoints of the range.

'`-T` *tempdir*'
'`--temporary-directory=`*tempdir*'

Use directory *tempdir* to store temporary files, overriding the `TMPDIR` environment variable. If this option is given more than once, temporary files are stored in all the directories given. If you have a large sort or merge that is I/O-bound, you can often improve performance by using this option to specify directories on different disks and controllers.

'`-u`'
'`--unique`'

Normally, output only the first of a sequence of lines that compare equal. For the '`--check`' ('`-c`') option, check that no pair of consecutive lines compares equal.

'`-k` *pos1* `[,`*pos2*`]`'
'`--key=`*pos1* `[,`*pos2*`]`'

Specify a sort field that consists of the part of the line between *pos1* and *pos2* (or the end of the line, if *pos2* is omitted), *inclusive*. Fields and character positions are numbered starting with 1. So to sort on the second field, you'd use '`--key=2,2`' ('`-k 2,2`'). See below for more examples.

'`-z`'
'`--zero-terminated`'

Treat the input as a set of lines, each terminated by a zero byte (ASCII NUL (Null) character) instead of an ASCII LF (Line Feed). This option can be useful in conjunction with '`perl -0`' or '`find -print0`' and '`xargs -0`' which do the same in order to reliably handle arbitrary pathnames (even those which contain Line Feed characters.)

Historical (BSD and System V) implementations of `sort` have differed in their interpretation of some options, particularly '`-b`', '`-f`', and '`-n`'. GNU sort follows the POSIX behavior, which is usually (but not always!) like the System V behavior. According to POSIX, '`-n`' no longer implies '`-b`'. For consistency, '`-M`' has been changed in the same way. This may affect the meaning of character positions in field specifications in obscure cases. The only fix is to add an explicit '`-b`'.

A position in a sort field specified with the '`-k`' option has the form '`f.c`', where *f* is the number of the field to use and *c* is the number of the first character from the beginning of the field. In a start position, an omitted '`.c`' stands for the field's first character. In an end position, an omitted or zero '.*c*' stands for the field's last character. If the '-b' option was specified, the '.*c*' part of a field specification is counted from the first nonblank character of the field.

A sort key position may also have any of the option letters 'Mbdfinr' appended to it, in which case the global ordering options are not used for that particular field. The '-b' option may be independently attached to either or both of the start and end positions of a field specification, and if it is inherited from the global options it will be attached to both. Keys may span multiple fields.

On older systems, sort supports an obsolete origin-zero syntax '+*pos1* [-*pos2*]' for specifying sort keys. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '-k' instead.

Here are some examples to illustrate various combinations of options.

- Sort in descending (reverse) numeric order.

```
sort -nr
```

- Sort alphabetically, omitting the first and second fields. This uses a single key composed of the characters beginning at the start of field three and extending to the end of each line.

```
sort -k 3
```

- Sort numerically on the second field and resolve ties by sorting alphabetically on the third and fourth characters of field five. Use ':' as the field delimiter.

```
sort -t : -k 2,2n -k 5.3,5.4
```

  Note that if you had written '-k 2' instead of '-k 2,2' sort would have used all characters beginning in the second field and extending to the end of the line as the primary *numeric* key. For the large majority of applications, treating keys spanning more than one field as numeric will not do what you expect.

  Also note that the 'n' modifier was applied to the field-end specifier for the first key. It would have been equivalent to specify '-k 2n,2' or '-k 2n,2n'. All modifiers except 'b' apply to the associated *field*, regardless of whether the modifier character is attached to the field-start and/or the field-end part of the key specifier.

- Sort the password file on the fifth field and ignore any leading white space. Sort lines with equal values in field five on the numeric user ID in field three.

```
sort -t : -k 5b,5 -k 3,3n /etc/passwd
```

  An alternative is to use the global numeric modifier '-n'.

```
sort -t : -n -k 5b,5 -k 3,3 /etc/passwd
```

- Generate a tags file in case-insensitive sorted order.

```
find src -type f -print0 | sort -t / -z -f | xargs -0 etags --append
```

  The use of '-print0', '-z', and '-0' in this case means that pathnames that contain Line Feed characters will not get broken up by the sort operation.

  Finally, to ignore both leading and trailing white space, you could have applied the 'b' modifier to the field-end specifier for the first key,

```
sort -t : -n -k 5b,5b -k 3,3 /etc/passwd
```

  or by using the global '-b' modifier instead of '-n' and an explicit 'n' with the second key specifier.

```
sort -t : -b -k 5,5 -k 3,3n /etc/passwd
```

7.2 `uniq`: Uniquify files

`uniq` writes the unique lines in the given '`input`', or standard input if nothing is given or for an *input* name of '`-`'. Synopsis:

```
uniq [option]... [input [output]]
```

By default, `uniq` prints the unique lines in a sorted file, i.e., discards all but one of identical successive lines. Optionally, it can instead show only lines that appear exactly once, or lines that appear more than once.

The input need not be sorted, but duplicate input lines are detected only if they are adjacent. If you want to discard non-adjacent duplicate lines, perhaps you want to use `sort -u`.

Comparisons use the character collating sequence specified by the `LC_COLLATE` locale category.

If no *output* file is specified, `uniq` writes to standard output.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-f n`'
'`--skip-fields=n`'
: Skip *n* fields on each line before checking for uniqueness. Fields are sequences of non-space non-tab characters that are separated from each other by at least one space or tab.

  On older systems, `uniq` supports an obsolete option '`-n`'. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '`-f n`' instead.

'`-s n`'
'`--skip-chars=n`'
: Skip *n* characters before checking for uniqueness. If you use both the field and character skipping options, fields are skipped over first.

  On older systems, `uniq` supports an obsolete option '`+n`'. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '`-s n`' instead.

'`-c`'
'`--count`'
: Print the number of times each line occurred along with the line.

'`-i`'
'`--ignore-case`'
: Ignore differences in case when comparing lines.

'`-d`'
'`--repeated`'
: Print one copy of each duplicate line.

'-D'
'--all-repeated[=*delimit-method*]'
Print all copies of each duplicate line. This option is useful mainly in conjunction with other options e.g., to ignore case or to compare only selected fields. The optional *delimit-method* tells how to delimit groups of duplicate lines, and must be one of the following:

'none' Do not delimit groups of duplicate lines. This is equivalent to '--all-repeated' ('-D').

'prepend' Output a newline before each group of duplicate lines.

'separate'
Separate groups of duplicate lines with a single newline. This is the same as using 'prepend', except that there is no newline before the first group, and hence may be better suited for output direct to users.

Note that when groups are delimited and the input stream contains two or more consecutive blank lines, then the output is ambiguous. To avoid that, filter the input through 'tr -s '\n'' to replace each sequence of consecutive newlines with a single newline.

This is a GNU extension.

'-u'
'--unique'
Print non-duplicate lines.

'-w *n*'
'--check-chars=*n*'
Compare *n* characters on each line (after skipping any specified fields and characters). By default the entire rest of the lines are compared.

7.3 comm: Compare two sorted files line by line

comm writes to standard output lines that are common, and lines that are unique, to two input files; a file name of '-' means standard input. Synopsis:

```
comm [option]... file1 file2
```

Before comm can be used, the input files must be sorted using the collating sequence specified by the LC_COLLATE locale. If an input file ends in a non-newline character, a newline is silently appended. The sort command with no options always outputs a file that is suitable input to comm.

With no options, comm produces three column output. Column one contains lines unique to *file1*, column two contains lines unique to *file2*, and column three contains lines common to both files. Columns are separated by a single TAB character.

The options '-1', '-2', and '-3' suppress printing of the corresponding columns. Also see Chapter 2 [Common options], page 2.

Unlike some other comparison utilities, `comm` has an exit status that does not depend on the result of the comparison. Upon normal completion `comm` produces an exit code of zero. If there is an error it exits with nonzero status.

7.4 `tsort`: Topological sort

`tsort` performs a topological sort on the given *file*, or standard input if no input file is given or for a *file* of '`-`'. For more details and some history, see Section 7.5 [tsort background], page 39. Synopsis:

```
tsort [option] [file]
```

`tsort` reads its input as pairs of strings, separated by blanks, indicating a partial ordering. The output is a total ordering that corresponds to the given partial ordering.

For example

```
tsort <<EOF
a b c
d
e f
b c d e
EOF
```

will produce the output

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

Consider a more realistic example. You have a large set of functions all in one file, and they may all be declared static except one. Currently that one (say `main`) is the first function defined in the file, and the ones it calls directly follow it, followed by those they call, etc. Let's say that you are determined to take advantage of prototypes, so you have to choose between declaring all of those functions (which means duplicating a lot of information from the definitions) and rearranging the functions so that as many as possible are defined before they are used. One way to automate the latter process is to get a list for each function of the functions it calls directly. Many programs can generate such lists. They describe a call graph. Consider the following list, in which a given line indicates that the function on the left calls the one on the right directly.

```
main parse_options
main tail_file
main tail_forever
tail_file pretty_name
tail_file write_header
tail_file tail
tail_forever recheck
tail_forever pretty_name
```

```
tail_forever write_header
tail_forever dump_remainder
tail tail_lines
tail tail_bytes
tail_lines start_lines
tail_lines dump_remainder
tail_lines file_lines
tail_lines pipe_lines
tail_bytes xlseek
tail_bytes start_bytes
tail_bytes dump_remainder
tail_bytes pipe_bytes
file_lines dump_remainder
recheck pretty_name
```

then you can use `tsort` to produce an ordering of those functions that satisfies your requirement.

```
example$ tsort call-graph | tac
dump_remainder
start_lines
file_lines
pipe_lines
xlseek
start_bytes
pipe_bytes
tail_lines
tail_bytes
pretty_name
write_header
tail
recheck
parse_options
tail_file
tail_forever
main
```

`tsort` detects any cycles in the input and writes the first cycle encountered to standard error.

Note that for a given partial ordering, generally there is no unique total ordering. In the context of the call graph above, the function `parse_options` may be placed anywhere in the list as long as it precedes `main`.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

7.5 tsort: Background

`tsort` exists because very early versions of the Unix linker processed an archive file exactly once, and in order. As `ld` read each object in the archive, it decided whether it was needed in the program based on whether it defined any symbols which were undefined at that point in the link.

This meant that dependencies within the archive had to be handled specially. For example, `scanf` probably calls `read`. That means that in a single pass through an archive, it was important for `scanf.o` to appear before read.o, because otherwise a program which calls `scanf` but not `read` might end up with an unexpected unresolved reference to `read`.

The way to address this problem was to first generate a set of dependencies of one object file on another. This was done by a shell script called `lorder`. The GNU tools don't provide a version of lorder, as far as I know, but you can still find it in BSD distributions.

Then you ran `tsort` over the `lorder` output, and you used the resulting sort to define the order in which you added objects to the archive.

This whole procedure has been obsolete since about 1980, because Unix archives now contain a symbol table (traditionally built by `ranlib`, now generally built by `ar` itself), and the Unix linker uses the symbol table to effectively make multiple passes over an archive file.

Anyhow, that's where tsort came from. To solve an old problem with the way the linker handled archive files, which has since been solved in different ways.

7.6 ptx: Produce permuted indexes

`ptx` reads a text file and essentially produces a permuted index, with each keyword in its context. The calling sketch is either one of:

```
ptx [option ...] [file ...]
ptx -G [option ...] [input [output]]
```

The '`-G`' (or its equivalent: '`--traditional`') option disables all GNU extensions and reverts to traditional mode, thus introducing some limitations and changing several of the program's default option values. When '`-G`' is not specified, GNU extensions are always enabled. GNU extensions to `ptx` are documented wherever appropriate in this document. For the full list, see See Section 7.6.5 [Compatibility in ptx], page 45.

Individual options are explained in the following sections.

When GNU extensions are enabled, there may be zero, one or several *file*s after the options. If there is no *file*, the program reads the standard input. If there is one or several *file*s, they give the name of input files which are all read in turn, as if all the input files were concatenated. However, there is a full contextual break between each file and, when automatic referencing is requested, file names and line numbers refer to individual text input files. In all cases, the program outputs the permuted index to the standard output.

When GNU extensions are *not* enabled, that is, when the program operates in traditional mode, there may be zero, one or two parameters besides the options. If there are no parameters, the program reads the standard input and outputs the permuted index to the standard output. If there is only one parameter, it names the text *input* to be read instead of the standard input. If two parameters are given, they give respectively the name of the *input* file to read and the name of the *output* file to produce. *Be very careful* to note that, in this case, the contents of file given by the second parameter is destroyed. This behavior is dictated by System V `ptx` compatibility; GNU Standards normally discourage output parameters not introduced by an option.

Note that for *any* file named as the value of an option or as an input text file, a single dash `-` may be used, in which case standard input is assumed. However, it would not make sense to use this convention more than once per program invocation.

7.6.1 General options

'`-C`'
'`--copyright`'
: Print a short note about the copyright and copying conditions, then exit without further processing.

'`-G`'
'`--traditional`'
: As already explained, this option disables all GNU extensions to `ptx` and switches to traditional mode.

'`--help`'
: Print a short help on standard output, then exit without further processing.

'`--version`'
: Print the program version on standard output, then exit without further processing.

7.6.2 Charset selection

As it is set up now, the program assumes that the input file is coded using 8-bit ISO 8859-1 code, also known as Latin-1 character set, *unless* it is compiled for MS-DOS, in which case it uses the character set of the IBM-PC. (GNU `ptx` is not known to work on smaller MS-DOS machines anymore.) Compared to 7-bit ASCII, the set of characters which are letters is different; this alters the behavior of regular expression matching. Thus, the default regular expression for a keyword allows foreign or diacriticized letters. Keyword sorting, however, is still crude; it obeys the underlying character set ordering quite blindly.

'`-f`'
'`--ignore-case`'
: Fold lower case letters to upper case for sorting.

7.6.3 Word selection and input processing

'`-b` *`file`*'

'`--break-file=`*`file`*'
: This option provides an alternative (to '`-W`') method of describing which characters make up words. It introduces the name of a file which contains a list of characters which can*not* be part of one word; this file is called the *Break file*.

Any character which is not part of the Break file is a word constituent. If both options '-b' and '-W' are specified, then '-W' has precedence and '-b' is ignored.

When GNU extensions are enabled, the only way to avoid newline as a break character is to write all the break characters in the file with no newline at all, not even at the end of the file. When GNU extensions are disabled, spaces, tabs and newlines are always considered as break characters even if not included in the Break file.

'-i *file*'
'--ignore-file=*file*'
: The file associated with this option contains a list of words which will never be taken as keywords in concordance output. It is called the *Ignore file*. The file contains exactly one word in each line; the end of line separation of words is not subject to the value of the '-S' option.

  There is a default Ignore file used by `ptx` when this option is not specified, usually found in '/usr/local/lib/eign' if this has not been changed at installation time. If you want to deactivate the default Ignore file, specify `/dev/null` instead.

'-o *file*'
'--only-file=*file*'
: The file associated with this option contains a list of words which will be retained in concordance output; any word not mentioned in this file is ignored. The file is called the *Only file*. The file contains exactly one word in each line; the end of line separation of words is not subject to the value of the '-S' option.

  There is no default for the Only file. When both an Only file and an Ignore file are specified, a word is considered a keyword only if it is listed in the Only file and not in the Ignore file.

'-r'
'--references'
: On each input line, the leading sequence of non-white space characters will be taken to be a reference that has the purpose of identifying this input line in the resulting permuted index. For more information about reference production, see See Section 7.6.4 [Output formatting in ptx], page 42. Using this option changes the default value for option '-S'.

  Using this option, the program does not try very hard to remove references from contexts in output, but it succeeds in doing so *when* the context ends exactly at the newline. If option '-r' is used with '-S' default value, or when GNU extensions are disabled, this condition is always met and references are completely excluded from the output contexts.

'-S *regexp*'
'--sentence-regexp=*regexp*'
: This option selects which regular expression will describe the end of a line or the end of a sentence. In fact, this regular expression is not the only distinction between end of lines or end of sentences, and input line boundaries have no special significance outside this option. By default, when GNU extensions are

enabled and if '-r' option is not used, end of sentences are used. In this case, this *regex* is imported from GNU Emacs:

```
[.?!][]\"')}]*\\($\\|\t\\|  \\)[ \t\n]*
```

Whenever GNU extensions are disabled or if '-r' option is used, end of lines are used; in this case, the default *regexp* is just:

```
\n
```

Using an empty *regexp* is equivalent to completely disabling end of line or end of sentence recognition. In this case, the whole file is considered to be a single big line or sentence. The user might want to disallow all truncation flag generation as well, through option '-F ""'. See section "Syntax of Regular Expressions" in *The GNU Emacs Manual*.

When the keywords happen to be near the beginning of the input line or sentence, this often creates an unused area at the beginning of the output context line; when the keywords happen to be near the end of the input line or sentence, this often creates an unused area at the end of the output context line. The program tries to fill those unused areas by wrapping around context in them; the tail of the input line or sentence is used to fill the unused area on the left of the output line; the head of the input line or sentence is used to fill the unused area on the right of the output line.

As a matter of convenience to the user, many usual backslashed escape sequences from the C language are recognized and converted to the corresponding characters by `ptx` itself.

'-W *regexp*'
'--word-regexp=*regexp*'

This option selects which regular expression will describe each keyword. By default, if GNU extensions are enabled, a word is a sequence of letters; the *regexp* used is '\w+'. When GNU extensions are disabled, a word is by default anything which ends with a space, a tab or a newline; the *regexp* used is '[^ \t\n]+'.

An empty *regexp* is equivalent to not using this option. See section "Syntax of Regular Expressions" in *The GNU Emacs Manual*.

As a matter of convenience to the user, many usual backslashed escape sequences, as found in the C language, are recognized and converted to the corresponding characters by `ptx` itself.

7.6.4 Output formatting

Output format is mainly controlled by the '-O' and '-T' options described in the table below. When neither '-O' nor '-T' are selected, and if GNU extensions are enabled, the program chooses an output format suitable for a dumb terminal. Each keyword occurrence is output to the center of one line, surrounded by its left and right contexts. Each field is properly justified, so the concordance output can be readily observed. As a special feature, if automatic references are selected by option '-A' and are output before the left context, that is, if option '-R' is *not* selected, then a colon is added after the reference; this nicely interfaces with GNU Emacs `next-error` processing. In this default output format, each white space character, like newline and tab, is merely changed to exactly one space, with no special attempt to compress consecutive spaces. This might change in the future. Except for those white space characters, every other character of the underlying set of 256 characters is transmitted verbatim.

Output format is further controlled by the following options.

'`-g` *number*'
'`--gap-size=`*number*'
: Select the size of the minimum white space gap between the fields on the output line.

'`-w` *number*'
'`--width=`*number*'
: Select the maximum output width of each final line. If references are used, they are included or excluded from the maximum output width depending on the value of option '`-R`'. If this option is not selected, that is, when references are output before the left context, the maximum output width takes into account the maximum length of all references. If this option is selected, that is, when references are output after the right context, the maximum output width does not take into account the space taken by references, nor the gap that precedes them.

'`-A`'
'`--auto-reference`'
: Select automatic references. Each input line will have an automatic reference made up of the file name and the line ordinal, with a single colon between them. However, the file name will be empty when standard input is being read. If both '`-A`' and '`-r`' are selected, then the input reference is still read and skipped, but the automatic reference is used at output time, overriding the input reference.

'`-R`'
'`--right-side-refs`'
: In the default output format, when option '`-R`' is not used, any references produced by the effect of options '`-r`' or '`-A`' are placed to the far right of output lines, after the right context. With default output format, when the '`-R`' option is specified, references are rather placed at the beginning of each output line, before the left context. For any other output format, option '`-R`' is ignored, with one exception: with '`-R`' the width of references is *not* taken into account in total output width given by '`-w`'.

  This option is automatically selected whenever GNU extensions are disabled.

'`-F` *string*'
'`--flac-truncation=`*string*'
: This option will request that any truncation in the output be reported using the string *string*. Most output fields theoretically extend towards the beginning or the end of the current line, or current sentence, as selected with option '`-S`'. But there is a maximum allowed output line width, changeable through option '`-w`', which is further divided into space for various output fields. When a field

has to be truncated because it cannot extend beyond the beginning or the end of the current line to fit in, then a truncation occurs. By default, the string used is a single slash, as in '`-F /`'.

string may have more than one character, as in '`-F ...`'. Also, in the particular case when *string* is empty ('`-F ""`'), truncation flagging is disabled, and no truncation marks are appended in this case.

As a matter of convenience to the user, many usual backslashed escape sequences, as found in the C language, are recognized and converted to the corresponding characters by `ptx` itself.

'`-M` *string*'
'`--macro-name=`*string*'
: Select another *string* to be used instead of '`xx`', while generating output suitable for `nroff`, `troff` or TEX.

'`-O`'
'`--format=roff`'
: Choose an output format suitable for `nroff` or `troff` processing. Each output line will look like:

```
.xx "tail" "before" "keyword_and_after" "head" "ref"
```

so it will be possible to write a '`.xx`' roff macro to take care of the output typesetting. This is the default output format when GNU extensions are disabled. Option '`-M`' can be used to change '`xx`' to another macro name.

In this output format, each non-graphical character, like newline and tab, is merely changed to exactly one space, with no special attempt to compress consecutive spaces. Each quote character: `"` is doubled so it will be correctly processed by `nroff` or `troff`.

'`-T`'
'`--format=tex`'
: Choose an output format suitable for TEX processing. Each output line will look like:

```
\xx {tail}{before}{keyword}{after}{head}{ref}
```

so it will be possible to write a `\xx` definition to take care of the output typesetting. Note that when references are not being produced, that is, neither option '`-A`' nor option '`-r`' is selected, the last parameter of each `\xx` call is inhibited. Option '`-M`' can be used to change '`xx`' to another macro name.

In this output format, some special characters, like `$`, `%`, `&`, `#` and `_` are automatically protected with a backslash. Curly brackets `{`, `}` are protected with a backslash and a pair of dollar signs (to force mathematical mode). The backslash itself produces the sequence `\backslash{}`. Circumflex and tilde diacritical marks produce the sequence `^\{ }` and `~\{ }` respectively. Other diacriticized characters of the underlying character set produce an appropriate TEX sequence as far as possible. The other non-graphical characters, like newline and tab, and all other characters which are not part of ASCII, are merely changed to exactly one space, with no special attempt to compress consecutive spaces. Let me know how to improve this special character processing for TEX.

7.6.5 The GNU extensions to ptx

This version of `ptx` contains a few features which do not exist in System V `ptx`. These extra features are suppressed by using the '`-G`' command line option, unless overridden by other command line options. Some GNU extensions cannot be recovered by overriding, so the simple rule is to avoid '`-G`' if you care about GNU extensions. Here are the differences between this program and System V `ptx`.

- This program can read many input files at once, it always writes the resulting concordance on standard output. On the other hand, System V `ptx` reads only one file and sends the result to standard output or, if a second *file* parameter is given on the command, to that *file*.

  Having output parameters not introduced by options is a dangerous practice which GNU avoids as far as possible. So, for using `ptx` portably between GNU and System V, you should always use it with a single input file, and always expect the result on standard output. You might also want to automatically configure in a '`-G`' option to `ptx` calls in products using `ptx`, if the configurator finds that the installed `ptx` accepts '`-G`'.
- The only options available in System V `ptx` are options '`-b`', '`-f`', '`-g`', '`-i`', '`-o`', '`-r`', '`-t`' and '`-w`'. All other options are GNU extensions and are not repeated in this enumeration. Moreover, some options have a slightly different meaning when GNU extensions are enabled, as explained below.
- By default, concordance output is not formatted for `troff` or `nroff`. It is rather formatted for a dumb terminal. `troff` or `nroff` output may still be selected through option '`-O`'.
- Unless '`-R`' option is used, the maximum reference width is subtracted from the total output line width. With GNU extensions disabled, width of references is not taken into account in the output line width computations.
- All 256 characters, even *NUL*s, are always read and processed from input file with no adverse effect, even if GNU extensions are disabled. However, System V `ptx` does not accept 8-bit characters, a few control characters are rejected, and the tilde `~` is also rejected.
- Input line length is only limited by available memory, even if GNU extensions are disabled. However, System V `ptx` processes only the first 200 characters in each line.
- The break (non-word) characters default to be every character except all letters of the underlying character set, diacriticized or not. When GNU extensions are disabled, the break characters default to space, tab and newline only.
- The program makes better use of output line width. If GNU extensions are disabled, the program rather tries to imitate System V `ptx`, but still, there are some slight disposition glitches this program does not completely reproduce.
- The user can specify both an Ignore file and an Only file. This is not allowed with System V `ptx`.

8 Operating on fields within a line

8.1 `cut`: Print selected parts of lines

`cut` writes to standard output selected parts of each line of each input file, or standard input if no files are given or for a file name of '`-`'. Synopsis:

```
cut [option]... [file]...
```

In the table which follows, the *byte-list*, *character-list*, and *field-list* are one or more numbers or ranges (two numbers separated by a dash) separated by commas. Bytes, characters, and fields are numbered starting at 1. Incomplete ranges may be given: '`-m`' means '`1-m`'; '`n-`' means '`n`' through end of line or last field. The list elements can be repeated, can overlap, and can be specified in any order; but the selected input is written in the same order that it is read, and is written exactly once.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-b byte-list`'
'`--bytes=byte-list`'
: Print only the bytes in positions listed in *byte-list*. Tabs and backspaces are treated like any other character; they take up 1 byte.

'`-c character-list`'
'`--characters=character-list`'
: Print only characters in positions listed in *character-list*. The same as '`-b`' for now, but internationalization will change that. Tabs and backspaces are treated like any other character; they take up 1 character.

'`-f field-list`'
'`--fields=field-list`'
: Print only the fields listed in *field-list*. Fields are separated by a TAB character by default. Also print any line that contains no delimiter character, unless the '`--only-delimited`' ('`-s`') option is specified

'`-d input_delim_byte`'
'`--delimiter=input_delim_byte`'
: For '`-f`', fields are separated in the input by the first character in *input_delim_byte* (default is TAB).

'`-n`'
: Do not split multi-byte characters (no-op for now).

'`-s`'
'`--only-delimited`'
: For '`-f`', do not print lines that do not contain the field separator character. Normally, any line without a field separator is printed verbatim.

'`--output-delimiter=output_delim_string`'
: For '`-f`', output fields are separated by *output_delim_string*. The default is to use the input delimiter.

8.2 paste: Merge lines of files

`paste` writes to standard output lines consisting of sequentially corresponding lines of each given file, separated by a TAB character. Standard input is used for a file name of '-' or if no input files are given.

For example:

```
$ cat num2
1
2
$ cat let3
a
b
c
$ paste num2 let3
1       a
2       b
        c
```

Synopsis:

```
paste [option]... [file]...
```

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-s'
'--serial'
: Paste the lines of one file at a time rather than one line from each file. Using the above example data:

```
$ paste -s num2 let3
1       2
a       b       c
```

'-d *delim-list*'
'--delimiters=*delim-list*'
: Consecutively use the characters in *delim-list* instead of TAB to separate merged lines. When *delim-list* is exhausted, start again at its beginning. Using the above example data:

```
$ paste -d '%_' num2 let3 num2
1%a_1
2%b_2
%c_
```

8.3 join: Join lines on a common field

`join` writes to standard output a line for each pair of input lines that have identical join fields. Synopsis:

```
join [option]... file1 file2
```

Either *file1* or *file2* (but not both) can be '`-`', meaning standard input. *file1* and *file2* should be sorted on the join fields.

Normally, the sort order is that of the collating sequence specified by the `LC_COLLATE` locale. Unless the '`-t`' option is given, the sort comparison ignores blanks at the start of the join field, as in `sort -b`. If the '`--ignore-case`' option is given, the sort comparison ignores the case of characters in the join field, as in `sort -f`.

However, as a GNU extension, if the input has no unpairable lines the sort order can be any order that considers two fields to be equal if and only if the sort comparison described above considers them to be equal. For example:

```
$ cat file1
a a1
c c1
b b1
$ cat file2
a a2
c c2
b b2
$ join file1 file2
a a1 a2
c c1 c2
b b1 b2
```

The defaults are: the join field is the first field in each line; fields in the input are separated by one or more blanks, with leading blanks on the line ignored; fields in the output are separated by a space; each output line consists of the join field, the remaining fields from *file1*, then the remaining fields from *file2*.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-a` *`file-number`*'
: Print a line for each unpairable line in file *file-number* (either '`1`' or '`2`'), in addition to the normal output.

'`-e` *`string`*'
: Replace those output fields that are missing in the input with *string*.

'`-i`'
'`--ignore-case`'
: Ignore differences in case when comparing keys. With this option, the lines of the input files must be ordered in the same way. Use '`sort -f`' to produce this ordering.

'`-1` *`field`*'
'`-j1` *`field`*'
: Join on field *field* (a positive integer) of file 1.

'`-2` *`field`*'
'`-j2` *`field`*'
: Join on field *field* (a positive integer) of file 2.

'-j *field*'
: Equivalent to '-1 *field* -2 *field*'.

'-o *field-list* ...'
: Construct each output line according to the format in *field-list*. Each element in *field-list* is either the single character '0' or has the form *m.n* where the file number, *m*, is '1' or '2' and *n* is a positive field number.

  A field specification of '0' denotes the join field. In most cases, the functionality of the '0' field spec may be reproduced using the explicit *m.n* that corresponds to the join field. However, when printing unpairable lines (using either of the '-a' or '-v' options), there is no way to specify the join field using *m.n* in *field-list* if there are unpairable lines in both files. To give `join` that functionality, POSIX invented the '0' field specification notation.

  The elements in *field-list* are separated by commas or blanks. Multiple *field-list* arguments can be given after a single '-o' option; the values of all lists given with '-o' are concatenated together. All output lines – including those printed because of any -a or -v option – are subject to the specified *field-list*.

'-t *char*'
: Use character *char* as the input and output field separator.

'-v *file-number*'
: Print a line for each unpairable line in file *file-number* (either '1' or '2'), instead of the normal output.

In addition, when GNU `join` is invoked with exactly one argument, the '--help' and '--version' options are recognized. See Chapter 2 [Common options], page 2.

9 Operating on characters

This commands operate on individual characters.

9.1 `tr`: Translate, squeeze, and/or delete characters

Synopsis:

```
tr [option]... set1 [set2]
```

`tr` copies standard input to standard output, performing one of the following operations:

- translate, and optionally squeeze repeated characters in the result,
- squeeze repeated characters,
- delete characters,
- delete characters, then squeeze repeated characters from the result.

The *set1* and (if given) *set2* arguments define ordered sets of characters, referred to below as *set1* and *set2*. These sets are the characters of the input that `tr` operates on. The '`--complement`' ('`-c`') option replaces *set1* with its complement (all of the characters that are not in *set1*).

9.1.1 Specifying sets of characters

The format of the *set1* and *set2* arguments resembles the format of regular expressions; however, they are not regular expressions, only lists of characters. Most characters simply represent themselves in these strings, but the strings can contain the shorthands listed below, for convenience. Some of them can be used only in *set1* or *set2*, as noted below.

Backslash escapes

A backslash followed by a character not listed below causes an error message.

'`\a`' Control-G.

'`\b`' Control-H.

'`\f`' Control-L.

'`\n`' Control-J.

'`\r`' Control-M.

'`\t`' Control-I.

'`\v`' Control-K.

'`\`*ooo*' The character with the value given by *ooo*, which is 1 to 3 octal digits,

'`\\`' A backslash.

Ranges

The notation '*`m-n`*' expands to all of the characters from *m* through *n*, in ascending order. *m* should collate before *n*; if it doesn't, an error results. As an example, '`0-9`' is the same as '`0123456789`'.

GNU `tr` does not support the System V syntax that uses square brackets to enclose ranges. Translations specified in that format sometimes work as expected, since the brackets are often transliterated to themselves. However, they should be avoided because they sometimes behave unexpectedly. For example, '`tr -d '[0-9]'`' deletes brackets as well as digits.

Many historically common and even accepted uses of ranges are not portable. For example, on EBCDIC hosts using the '`A-Z`' range will not do what most would expect because '`A`' through '`Z`' are not contiguous as they are in ASCII. If you can rely on a POSIX compliant version of `tr`, then the best way to work around this is to use character classes (see below). Otherwise, it is most portable (and most ugly) to enumerate the members of the ranges.

Repeated characters

The notation '`[c*n]`' in *set2* expands to *n* copies of character *c*. Thus, '`[y*6]`' is the same as '`yyyyyy`'. The notation '`[c*]`' in *string2* expands to as many copies of *c* as are needed to make *set2* as long as *set1*. If *n* begins with '`0`', it is interpreted in octal, otherwise in decimal.

Character classes

The notation '`[:class:]`' expands to all of the characters in the (predefined) class *class*. The characters expand in no particular order, except for the `upper` and `lower` classes, which expand in ascending order. When the '`--delete`' ('`-d`') and '`--squeeze-repeats`' ('`-s`') options are both given, any character class can be used in *set2*. Otherwise, only the character classes `lower` and `upper` are accepted in *set2*, and then only if the corresponding character class (`upper` and `lower`, respectively) is specified in the same relative position in *set1*. Doing this specifies case conversion. The class names are given below; an error results when an invalid class name is given.

`alnum` Letters and digits.

`alpha` Letters.

`blank` Horizontal whitespace.

`cntrl` Control characters.

`digit` Digits.

`graph` Printable characters, not including space.

`lower` Lowercase letters.

`print` Printable characters, including space.

`punct` Punctuation characters.

`space` Horizontal or vertical whitespace.

`upper` Uppercase letters.

`xdigit` Hexadecimal digits.

Equivalence classes

The syntax '`[=c=]`' expands to all of the characters that are equivalent to *c*, in no particular order. Equivalence classes are a relatively recent invention intended to support non-English alphabets. But there seems to be no standard way to define them or determine their contents. Therefore, they are not fully implemented in GNU `tr`; each character's equivalence class consists only of that character, which is of no particular use.

9.1.2 Translating

`tr` performs translation when *set1* and *set2* are both given and the '`--delete`' ('`-d`') option is not given. `tr` translates each character of its input that is in *set1* to the corresponding character in *set2*. Characters not in *set1* are passed through unchanged. When a character appears more than once in *set1* and the corresponding characters in *set2* are not all the same, only the final one is used. For example, these two commands are equivalent:

```
tr aaa xyz
tr a z
```

A common use of `tr` is to convert lowercase characters to uppercase. This can be done in many ways. Here are three of them:

```
tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
tr a-z A-Z
tr '[:lower:]' '[:upper:]'
```

But note that using ranges like `a-z` above is not portable.

When `tr` is performing translation, *set1* and *set2* typically have the same length. If *set1* is shorter than *set2*, the extra characters at the end of *set2* are ignored.

On the other hand, making *set1* longer than *set2* is not portable; POSIX says that the result is undefined. In this situation, BSD `tr` pads *set2* to the length of *set1* by repeating the last character of *set2* as many times as necessary. System V `tr` truncates *set1* to the length of *set2*.

By default, GNU `tr` handles this case like BSD `tr`. When the '`--truncate-set1`' ('`-t`') option is given, GNU `tr` handles this case like the System V `tr` instead. This option is ignored for operations other than translation.

Acting like System V `tr` in this case breaks the relatively common BSD idiom:

```
tr -cs A-Za-z0-9 '\012'
```

because it converts only zero bytes (the first element in the complement of *set1*), rather than all non-alphanumerics, to newlines.

By the way, the above idiom is not portable because it uses ranges. Assuming a POSIX compliant `tr`, here is a better way to write it:

```
tr -cs '[:alnum:]' '[\n*]'
```

9.1.3 Squeezing repeats and deleting

When given just the '`--delete`' ('`-d`') option, `tr` removes any input characters that are in *set1*.

When given just the '`--squeeze-repeats`' ('`-s`') option, `tr` replaces each input sequence of a repeated character that is in *set1* with a single occurrence of that character.

When given both '`--delete`' and '`--squeeze-repeats`', `tr` first performs any deletions using *set1*, then squeezes repeats from any remaining characters using *set2*.

The '`--squeeze-repeats`' option may also be used when translating, in which case `tr` first performs translation, then squeezes repeats from any remaining characters using *set2*.

Here are some examples to illustrate various combinations of options:

- Remove all zero bytes:

  ```
  tr -d '\000'
  ```

- Put all words on lines by themselves. This converts all non-alphanumeric characters to newlines, then squeezes each string of repeated newlines into a single newline:

  ```
  tr -cs '[:alnum:]' '[\n*]'
  ```

- Convert each sequence of repeated newlines to a single newline:

  ```
  tr -s '\n'
  ```

- Find doubled occurrences of words in a document. For example, people often write "the the" with the duplicated words separated by a newline. The bourne shell script below works first by converting each sequence of punctuation and blank characters to a single newline. That puts each "word" on a line by itself. Next it maps all uppercase characters to lower case, and finally it runs `uniq` with the '`-d`' option to print out only the words that were adjacent duplicates.

  ```
  #!/bin/sh
  cat "$@" \
    | tr -s '[:punct:][:blank:]' '\n' \
    | tr '[:upper:]' '[:lower:]' \
    | uniq -d
  ```

- Deleting a small set of characters is usually straightforward. For example, to remove all '`a`'s, '`x`'s, and '`M`'s you would do this:

  ```
  tr -d axM
  ```

  However, when '`-`' is one of those characters, it can be tricky because '`-`' has special meanings. Performing the same task as above but also removing all '`-`' characters, we might try `tr -d -axM`, but that would fail because `tr` would try to interpret '`-a`' as a command-line option. Alternatively, we could try putting the hyphen inside the string, `tr -d a-xM`, but that wouldn't work either because it would make `tr` interpret `a-x` as the range of characters '`a`'...'`x`' rather than the three. One way to solve the problem is to put the hyphen at the end of the list of characters:

  ```
  tr -d axM-
  ```

  More generally, use the character class notation `[=c=]` with '`-`' (or any other character) in place of the '`c`':

  ```
  tr -d '[=-=]axM'
  ```

  Note how single quotes are used in the above example to protect the square brackets from interpretation by a shell.

9.1.4 Warning messages

Setting the environment variable `POSIXLY_CORRECT` turns off the following warning and error messages, for strict compliance with POSIX. Otherwise, the following diagnostics are issued:

1. When the '`--delete`' option is given but '`--squeeze-repeats`' is not, and *set2* is given, GNU `tr` by default prints a usage message and exits, because *set2* would not be used. The POSIX specification says that *set2* must be ignored in this case. Silently ignoring arguments is a bad idea.
2. When an ambiguous octal escape is given. For example, '`\400`' is actually '`\40`' followed by the digit '`0`', because the value 400 octal does not fit into a single byte.

GNU `tr` does not provide complete BSD or System V compatibility. For example, it is impossible to disable interpretation of the POSIX constructs '`[:alpha:]`', '`[=c=]`', and '`[c*10]`'. Also, GNU `tr` does not delete zero bytes automatically, unlike traditional Unix versions, which provide no way to preserve zero bytes.

9.2 `expand`: Convert tabs to spaces

`expand` writes the contents of each given *file*, or standard input if none are given or for a *file* of '`-`', to standard output, with tab characters converted to the appropriate number of spaces. Synopsis:

```
expand [option]... [file]...
```

By default, `expand` converts all tabs to spaces. It preserves backspace characters in the output; they decrement the column count for tab calculations. The default action is equivalent to '`-t 8`' (set tabs every 8 columns).

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-t` *tab1*`[,`*tab2*`]...`'
'`--tabs=`*tab1*`[,`*tab2*`]...`'

: If only one tab stop is given, set the tabs *tab1* spaces apart (default is 8). Otherwise, set the tabs at columns *tab1*, *tab2*, ... (numbered from 0), and replace any tabs beyond the last tabstop given with single spaces. Tabstops can be separated by blanks as well as by commas.

: On older systems, `expand` supports an obsolete option '`-`*tab1*`[,`*tab2*`]...`', where tabstops must be separated by commas. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '`-t` *tab1*`[,`*tab2*`]...`' instead.

'`-i`'
'`--initial`'

: Only convert initial tabs (those that precede all non-space or non-tab characters) on each line to spaces.

9.3 `unexpand`: Convert spaces to tabs

`unexpand` writes the contents of each given *file*, or standard input if none are given or for a *file* of '-', to standard output, with strings of two or more space or tab characters converted to as many tabs as possible followed by as many spaces as are needed. Synopsis:

```
unexpand [option]... [file]...
```

By default, `unexpand` converts only initial spaces and tabs (those that precede all non space or tab characters) on each line. It preserves backspace characters in the output; they decrement the column count for tab calculations. By default, tabs are set at every 8th column.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-t` *tab1*`[,`*tab2*`]...`'
'`--tabs=`*tab1*`[,`*tab2*`]...`'
: If only one tab stop is given, set the tabs *tab1* spaces apart instead of the default 8. Otherwise, set the tabs at columns *tab1*, *tab2*, ... (numbered from 0), and leave spaces and tabs beyond the tabstops given unchanged. Tabstops can be separated by blanks as well as by commas. This option implies the '`-a`' option.

  On older systems, `unexpand` supports an obsolete option '`-`*tab1*`[,`*tab2*`]...`', where tabstops must be separated by commas. (Unlike '`-t`', this obsolete option does not imply '`-a`'.) POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '`--first-only -t` *tab1*`[,`*tab2*`]...`' instead.

'`-a`'
'`--all`'
: Convert all strings of two or more spaces or tabs, not just initial ones, to tabs.

10 Directory listing

This chapter describes the `ls` command and its variants `dir` and `vdir`, which list information about files.

10.1 `ls`: List directory contents

The `ls` program lists information about files (of any type, including directories). Options and file arguments can be intermixed arbitrarily, as usual.

For non-option command-line arguments that are directories, by default `ls` lists the contents of directories, not recursively, and omitting files with names beginning with '`.`'. For other non-option arguments, by default `ls` lists just the file name. If no non-option argument is specified, `ls` operates on the current directory, acting as if it had been invoked with a single argument of '`.`'.

By default, the output is sorted alphabetically, according to the locale settings in effect.[1] If standard output is a terminal, the output is in columns (sorted vertically) and control characters are output as question marks; otherwise, the output is listed one per line and control characters are output as-is.

Because `ls` is such a fundamental program, it has accumulated many options over the years. They are described in the subsections below; within each section, options are listed alphabetically (ignoring case). The division of options into the subsections is not absolute, since some options affect more than one aspect of `ls`'s operation.

Also see Chapter 2 [Common options], page 2.

10.1.1 Which files are listed

These options determine which files `ls` lists information for. By default, any files and the contents of any directories on the command line are shown.

'`-a`'
'`--all`' List all files in directories, including files that start with '`.`'.

'`-A`'
'`--almost-all`'
: List all files in directories except for '`.`' and '`..`'.

'`-B`'
'`--ignore-backups`'
: Do not list files that end with '`~`', unless they are given on the command line.

'`-d`'
'`--directory`'
: List just the names of directories, as with other types of files, rather than listing their contents. Do not follow symbolic links listed on the command line unless the '`--dereference-command-line`' ('`-H`') or '`--dereference`' ('`-L`') options are specified.

---

[1] If you use a non-POSIX locale (e.g., by setting `LC_ALL` to '`en_US`'), then `ls` may produce output that is sorted differently than you're accustomed to. In that case, set the `LC_ALL` environment variable to '`C`'.

'-H'
'--dereference-command-line'
: If a command line argument specifies a symbolic link, show information for the file the link references rather than for the link itself.

'-I PATTERN'
'--ignore=PATTERN'
: Do not list files whose names match the shell pattern (not regular expression) *pattern* unless they are given on the command line. As in the shell, an initial '.' in a file name does not match a wildcard at the start of *pattern*. Sometimes it is useful to give this option several times. For example,

```
$ ls --ignore='.??*' --ignore='.[^.]' --ignore='#*'
```

The first option ignores names of length 3 or more that start with '.', the second ignores all two-character names that start with '.' except '..', and the third ignores names that start with '#'.

'-L'
'--dereference'
: When showing file information for a symbolic link, show information for the file the link references rather than the link itself.

'-R'
'--recursive'
: List the contents of all directories recursively.

10.1.2 What information is listed

These options affect the information that `ls` displays. By default, only file names are shown.

'--author'
: List each file's author when producing long format directory listings. In GNU/Hurd, file authors can differ from their owners, but in other operating systems the two are the same.

'-D'
'--dired'
: With the long listing ('-l') format, print an additional line after the main output:

```
//DIRED// beg1 end1 beg2 end2 ...
```

The *begN* and *endN* are unsigned integers that record the byte position of the beginning and end of each file name in the output. This makes it easy for Emacs to find the names, even when they contain unusual characters such as space or newline, without fancy searching.

If directories are being listed recursively (`-R`), output a similar line with offsets for each subdirectory name:

```
//SUBDIRED// beg1 end1 ...
```

Finally, output a line of the form:

```
//DIRED-OPTIONS// --quoting-style=word
```

where *word* is the quoting style (see Section 10.1.7 [Formatting the file names], page 65).

Here is an actual example:

```
$ mkdir -p a/sub/deeper a/sub2
$ touch a/f1 a/f2
$ touch a/sub/deeper/file
$ ls -gloRF --dired a
  a:
  total 8
  -rw-r--r--    1        0 Nov  9 18:30 f1
  -rw-r--r--    1        0 Nov  9 18:30 f2
  drwxr-xr-x    3     4096 Nov  9 18:30 sub/
  drwxr-xr-x    2     4096 Nov  9 18:30 sub2/

  a/sub:
  total 4
  drwxr-xr-x    2     4096 Nov  9 18:30 deeper/

  a/sub/deeper:
  total 0
  -rw-r--r--    1        0 Nov  9 18:30 file

  a/sub2:
  total 0
//DIRED// 55 57 98 100 141 144 186 190 252 258 327 331
//SUBDIRED// 2 3 195 200 263 275 335 341
//DIRED-OPTIONS// --quoting-style=literal
```

Note that the pairs of offsets on the '`//DIRED//`' line above delimit these names: '`f1`', '`f2`', '`sub`', '`sub2`', '`deeper`', '`file`'. The offsets on the '`//SUBDIRED//`' line delimit the following directory names: '`a`', '`a/sub`', '`a/sub/deeper`', '`a/sub2`'.

Here is an example of how to extract the fifth entry name, '`deeper`', corresponding to the pair of offsets, 252 and 258:

```
$ ls -gloRF --dired a > out
$ dd bs=1 skip=252 count=6 < out 2>/dev/null; echo
deeper
```

Note that although the listing above includes a trailing slash for the '`deeper`' entry, the offsets select the name without the trailing slash. However, if you invoke `ls` with '`--dired`' along with an option like '`--escape`' (aka '`-b`') and operate on a file whose name contains special characters, notice that the backslash *is* included:

```
$ touch 'a b'
$ ls -blog --dired 'a b'
  -rw-r--r--    1        0 Nov  9 18:41 a\ b
//DIRED// 40 44
```

```
//DIRED-OPTIONS// --quoting-style=escape
```

If you use a quoting style that adds quote marks (e.g., '`--quoting-style=c`'), then the offsets include the quote marks. So beware that the user may select the quoting style via the environment variable `QUOTING_STYLE`. Hence, applications using '`--dired`' should either specify an explicit '`--quoting-style=literal`' option (aka '`-N`' or '`--literal`') on the command line, or else be prepared to parse the escaped names.

'`--full-time`'
: Produce long format directory listings, and list times in full. It is equivalent to using '`--format=long`' with '`--time-style=full-iso`' (see Section 10.1.6 [Formatting file timestamps], page 64).

'`-g`'
: Produce long format directory listings, but don't display owner information.

'`-G`'
'`--no-group`'
: Inhibit display of group information in a long format directory listing. (This is the default in some non-GNU versions of `ls`, so we provide this option for compatibility.)

'`-h`'
'`--human-readable`'
: Append a size letter to each size, such as '`M`' for mebibytes. Powers of 1024 are used, not 1000; '`M`' stands for 1,048,576 bytes. This option is equivalent to '`--block-size=human`' (see Section 2.2 [Block size], page 3). Use the '`--si`' option if you prefer powers of 1000.

'`-i`'
'`--inode`'
: Print the inode number (also called the file serial number and index number) of each file to the left of the file name. (This number uniquely identifies each file within a particular filesystem.)

'`-l`'
'`--format=long`'
'`--format=verbose`'
: In addition to the name of each file, print the file type, permissions, number of hard links, owner name, group name, size, and timestamp (see Section 10.1.6 [Formatting file timestamps], page 64), normally the modification time.

Normally the size is printed as a byte count without punctuation, but this can be overridden (see Section 2.2 [Block size], page 3). For example, '`-h`' prints an abbreviated, human-readable count, and '`--block-size="'1"`' prints a byte count with the thousands separator of the current locale.

For each directory that is listed, preface the files with a line '`total` *`blocks`*', where *blocks* is the total disk allocation for all files in that directory. The block size currently defaults to 1024 bytes, but this can be overridden (see Section 2.2 [Block size], page 3). The *blocks* computed counts each hard link separately; this is arguably a deficiency.

The permissions listed are similar to symbolic mode specifications (see Section 26.2 [Symbolic Modes], page 135). But `ls` combines multiple bits into the third character of each set of permissions as follows:

'`s`' If the setuid or setgid bit and the corresponding executable bit are both set.

'`S`' If the setuid or setgid bit is set but the corresponding executable bit is not set.

'`t`' If the sticky bit and the other-executable bit are both set.

'`T`' If the sticky bit is set but the other-executable bit is not set.

'`x`' If the executable bit is set and none of the above apply.

'`-`' Otherwise.

Following the permission bits is a single character that specifies whether an alternate access method applies to the file. When that character is a space, there is no alternate access method. When it is a printing character (e.g., '`+`'), then there is such a method.

'`-n`'
'`--numeric-uid-gid`'
Produce long format directory listings, but display numeric UIDs and GIDs instead of the owner and group names.

'`-o`' Produce long format directory listings, but don't display group information. It is equivalent to using '`--format=long`' with '`--no-group`' .

'`-s`'
'`--size`' Print the disk allocation of each file to the left of the file name. This is the amount of disk space used by the file, which is usually a bit more than the file's size, but it can be less if the file has holes.

Normally the disk allocation is printed in units of 1024 bytes, but this can be overridden (see Section 2.2 [Block size], page 3).

For files that are NFS-mounted from an HP-UX system to a BSD system, this option reports sizes that are half the correct values. On HP-UX systems, it reports sizes that are twice the correct values for files that are NFS-mounted from BSD systems. This is due to a flaw in HP-UX; it also affects the HP-UX `ls` program.

'`--si`' Append an SI-style abbreviation to each size, such as '`MB`' for megabytes. Powers of 1000 are used, not 1024; '`MB`' stands for 1,000,000 bytes. This option is equivalent to '`--block-size=si`'. Use the '`-h`' or '`--human-readable`' option if you prefer powers of 1024.

10.1.3 Sorting the output

These options change the order in which `ls` sorts the information it outputs. By default, sorting is done by character code (e.g., ASCII order).

'-c'
'--time=ctime'
'--time=status'
'--time=use'
: If the long listing format (e.g., '-l', '-o') is being used, print the status change time (the 'ctime' in the inode) instead of the modification time. When explicitly sorting by time ('--sort=time' or '-t') or when not using a long listing format, sort according to the status change time.

'-f' Primarily, like '-U'—do not sort; list the files in whatever order they are stored in the directory. But also enable '-a' (list all files) and disable '-l', '--color', and '-s' (if they were specified before the '-f').

'-r'
'--reverse'
: Reverse whatever the sorting method is—e.g., list files in reverse alphabetical order, youngest first, smallest first, or whatever.

'-S'
'--sort=size'
: Sort by file size, largest first.

'-t'
'--sort=time'
: Sort by modification time (the 'mtime' in the inode), newest first.

'-u'
'--time=atime'
'--time=access'
: If the long listing format (e.g., '--format=long') is being used, print the last access time (the 'atime' in the inode). When explicitly sorting by time ('--sort=time' or '-t') or when not using a long listing format, sort according to the access time.

'-U'
'--sort=none'
: Do not sort; list the files in whatever order they are stored in the directory. (Do not do any of the other unrelated things that '-f' does.) This is especially useful when listing very large directories, since not doing any sorting can be noticeably faster.

'-v'
'--sort=version'
: Sort by version name and number, lowest first. It behaves like a default sort, except that each sequence of decimal digits is treated numerically as an index/version number. (See Section 10.1.4 [More details about version sort], page 62.)

'-X'
'--sort=extension'
: Sort directory contents alphabetically by file extension (characters after the last '.'); files with no extension are sorted first.

10.1.4 More details about version sort

The version sort takes into account the fact that file names frequently include indices or version numbers. Standard sorting functions usually do not produce the ordering that people expect because comparisons are made on a character-by-character basis. The version sort addresses this problem, and is especially useful when browsing directories that contain many files with indices/version numbers in their names:

```
> ls -1             > ls -1v
foo.zml-1.gz        foo.zml-1.gz
foo.zml-100.gz      foo.zml-2.gz
foo.zml-12.gz       foo.zml-6.gz
foo.zml-13.gz       foo.zml-12.gz
foo.zml-2.gz        foo.zml-13.gz
foo.zml-25.gz       foo.zml-25.gz
foo.zml-6.gz        foo.zml-100.gz
```

Note also that numeric parts with leading zeroes are considered as fractional one:

```
> ls -1             > ls -1v
abc-1.007.tgz       abc-1.007.tgz
abc-1.012b.tgz      abc-1.01a.tgz
abc-1.01a.tgz       abc-1.012b.tgz
```

10.1.5 General output formatting

These options affect the appearance of the overall output.

'-1'
'--format=single-column'
: List one file per line. This is the default for `ls` when standard output is not a terminal.

'-C'
'--format=vertical'
: List files in columns, sorted vertically. This is the default for `ls` if standard output is a terminal. It is always the default for the `dir` and `d` programs. GNU `ls` uses variable width columns to display as many files as possible in the fewest lines.

'--color [=*when*]'
: Specify whether to use color for distinguishing file types. *when* may be omitted, or one of:

- none - Do not use color at all. This is the default.
- auto - Only use color if standard output is a terminal.
- always - Always use color.

Specifying '--color' and no *when* is equivalent to '--color=always'. Piping a colorized listing through a pager like `more` or `less` usually produces unreadable results. However, using `more -f` does seem to work.

'-F'
'--classify'
'--indicator-style=classify'

Append a character to each file name indicating the file type. Also, for regular files that are executable, append '*'. The file type indicators are '/' for directories, '@' for symbolic links, '|' for FIFOs, '=' for sockets, and nothing for regular files. Do not follow symbolic links listed on the command line unless the '--dereference-command-line' ('-H') or '--dereference' ('-L') options are specified.

'--indicator-style=*word*'

Append a character indicator with style *word* to entry names, as follows:

'none' Do not append any character indicator; this is the default.

'file-type'

Append '/' for directories, '@' for symbolic links, '|' for FIFOs, '=' for sockets, and nothing for regular files. This is the same as the '-p' or '--file-type' option.

'classify'

Append '*' for executable regular files, otherwise behave as for 'file-type'. This is the same as the '-F' or '--classify' option.

'-k' Print file sizes in 1024-byte blocks, overriding the default block size (see Section 2.2 [Block size], page 3). This option is equivalent to '--block-size=1K'.

'-m'
'--format=commas'

List files horizontally, with as many as will fit on each line, separated by ', ' (a comma and a space).

'-p'
'--file-type'
'--indicator-style=file-type'

Append a character to each file name indicating the file type. This is like '-F', except that executables are not marked.

'-x *format*'
'--format=across'
'--format=horizontal'

List the files in columns, sorted horizontally.

'-T *cols*'
'--tabsize=*cols*'

Assume that each tabstop is *cols* columns wide. The default is 8. ls uses tabs where possible in the output, for efficiency. If *cols* is zero, do not use tabs at all.

'-w'
'--width=*cols*'

Assume the screen is *cols* columns wide. The default is taken from the terminal settings if possible; otherwise the environment variable `COLUMNS` is used if it is set; otherwise the default is 80.

10.1.6 Formatting file timestamps

By default, file timestamps are listed in abbreviated form. Most locales use a timestamp like '`2002-03-30 23:45`'. However, the default POSIX locale uses a date like '`Mar 30  2002`' for non-recent timestamps, and a date-without-year and time like '`Mar 30 23:45`' for recent timestamps.

A timestamp is considered to be *recent* if it is less than six months old, and is not dated in the future. If a timestamp dated today is not listed in recent form, the timestamp is in the future, which means you probably have clock skew problems which may break programs like `make` that rely on file timestamps.

The following option changes how file timestamps are printed.

'--time-style=*style*'

List timestamps in style *style*. The *style* should be one of the following:

'+*format*'
List timestamps using *format*, where *format* is interpreted like the format argument of `date` (see Section 21.1 [date invocation], page 117). For example, '`--time-style="+%Y-%m-%d %H:%M:%S"`' causes `ls` to list timestamps like '`2002-03-30 23:45:56`'. As with `date`, *format*'s interpretation is affected by the `LC_TIME` locale category.

If *format* contains two format strings separated by a newline, the former is used for non-recent files and the latter for recent files; if you want output columns to line up, you may need to insert spaces in one of the two formats.

'full-iso'

List timestamps in full using ISO 8601 date, time, and time zone format with nanosecond precision, e.g., '`2002-03-30 23:45:56.477817180 -0700`'. This style is equivalent to '`+%Y-%m-%d %H:%M:%S.%N %z`'.

This is useful because the time output includes all the information that is available from the operating system. For example, this can help explain `make`'s behavior, since GNU `make` uses the full timestamp to determine whether a file is out of date.

'long-iso'

List ISO 8601 date and time in minutes, e.g., '`2002-03-30 23:45`'. These timestamps are shorter than '`full-iso`' timestamps, and are usually good enough for everyday work. This style is equivalent to '`%Y-%m-%d %H:%M`'.

'iso' List ISO 8601 dates for non-recent timestamps (e.g., '2002-03-30 '), and ISO 8601 month, day, hour, and minute for recent timestamps (e.g., '03-30 23:45'). These timestamps are uglier than 'long-iso' timestamps, but they carry nearly the same information in a smaller space and their brevity helps `ls` output fit within traditional 80-column output lines. The following two `ls` invocations are equivalent:

```
newline='
'
ls -l --time-style="+%Y-%m-%d $newline%m-%d %H:%M"
ls -l --time-style="iso"
```

'locale' List timestamps in a locale-dependent form. For example, a Finnish locale might list non-recent timestamps like 'maalis 30 2002' and recent timestamps like 'maalis 30 23:45'. Locale-dependent timestamps typically consume more space than 'iso' timestamps and are harder for programs to parse because locale conventions vary so widely, but they are easier for many people to read.

The `LC_TIME` locale category specifies the timestamp format. The default POSIX locale uses timestamps like 'Mar 30 2002' and 'Mar 30 23:45'; in this locale, the following two `ls` invocations are equivalent:

```
newline='
'
ls -l --time-style="+%b %e  %Y$newline%b %e %H:%M"
ls -l --time-style="locale"
```

Other locales behave differently. For example, in a German locale, '--time-style="locale"' might be equivalent to '--time-style="+%e. %b %Y $newline%e. %b %H:%M"' and might generate timestamps like '30. Mär 2002 ' and '30. Mär 23:45'.

'posix-*style*'
List POSIX-locale timestamps if the `LC_TIME` locale category is POSIX, *style* timestamps otherwise. For example, the default style, which is 'posix-long-iso', lists timestamps like 'Mar 30 2002' and 'Mar 30 23:45' when in the POSIX locale, and like '2002-03-30 23:45' otherwise.

You can specify the default value of the '--time-style' option with the environment variable `TIME_STYLE`; if `TIME_STYLE` is not set the default style is 'posix-long-iso'. GNU Emacs 21 and later can parse ISO dates, but older Emacs versions do not, so if you are using an older version of Emacs and specify a non-POSIX locale, you may need to set 'TIME_STYLE="locale"'.

10.1.7 Formatting the file names

These options change how file names themselves are printed.

'-b'
'--escape'
'--quoting-style=escape'
: Quote nongraphic characters in file names using alphabetic and octal backslash sequences like those used in C.

'-N'
'--literal'
'--quoting-style=literal'
: Do not quote file names.

'-q'
'--hide-control-chars'
: Print question marks instead of nongraphic characters in file names. This is the default if the output is a terminal and the program is `ls`.

'-Q'
'--quote-name'
'--quoting-style=c'
: Enclose file names in double quotes and quote nongraphic characters as in C.

'--quoting-style=*word*'
: Use style *word* to quote output names. The *word* should be one of the following:

  'literal' Output names as-is; this is the same as the '-N' or '--literal' option.

  'shell' Quote names for the shell if they contain shell metacharacters or would cause ambiguous output.

  'shell-always'
  Quote names for the shell, even if they would normally not require quoting.

  'c' Quote names as for a C language string; this is the same as the '-Q' or '--quote-name' option.

  'escape' Quote as with 'c' except omit the surrounding double-quote characters; this is the same as the '-b' or '--escape' option.

  'clocale' Quote as with 'c' except use quotation marks appropriate for the locale.

  'locale' Like 'clocale', but quote `'like this'` instead of `"like this"` in the default C locale. This looks nicer on many displays.

  You can specify the default value of the '--quoting-style' option with the environment variable `QUOTING_STYLE`. If that environment variable is not set, the default value is 'literal', but this default may change to 'shell' in a future version of this package.

'--show-control-chars'
: Print nongraphic characters as-is in file names. This is the default unless the output is a terminal and the program is `ls`.

10.2 `dir`: Briefly list directory contents

`dir` (also installed as `d`) is equivalent to `ls -C -b`; that is, by default files are listed in columns, sorted vertically, and special characters are represented by backslash escape sequences.

See Section 10.1 [ls invocation], page 56.

10.3 `vdir`: Verbosely list directory contents

`vdir` (also installed as `v`) is equivalent to `ls -l -b`; that is, by default files are listed in long format and special characters are represented by backslash escape sequences.

10.4 `dircolors`: Color setup for `ls`

`dircolors` outputs a sequence of shell commands to set up the terminal for color output from `ls` (and `dir`, etc.). Typical usage:

```
eval `dircolors [option]... [file]`
```

If *file* is specified, `dircolors` reads it to determine which colors to use for which file types and extensions. Otherwise, a precompiled database is used. For details on the format of these files, run '`dircolors --print-database`'.

The output is a shell command to set the `LS_COLORS` environment variable. You can specify the shell syntax to use on the command line, or `dircolors` will guess it from the value of the `SHELL` environment variable.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-b`'
'`--sh`'
'`--bourne-shell`'
: Output Bourne shell commands. This is the default if the `SHELL` environment variable is set and does not end with '`csh`' or '`tcsh`'.

'`-c`'
'`--csh`'
'`--c-shell`'
: Output C shell commands. This is the default if `SHELL` ends with `csh` or `tcsh`.

'`-p`'
'`--print-database`'
: Print the (compiled-in) default color configuration database. This output is itself a valid configuration file, and is fairly descriptive of the possibilities.

11 Basic operations

This chapter describes the commands for basic file manipulation: copying, moving (renaming), and deleting (removing).

11.1 `cp`: Copy files and directories

`cp` copies files (or, optionally, directories). The copy is completely independent of the original. You can either copy one file to another, or copy arbitrarily many files to a destination directory. Synopsis:

```
cp [option]... source dest
cp [option]... source... directory
```

If the last argument names an existing directory, `cp` copies each *source* file into that directory (retaining the same name). Otherwise, if only two files are given, it copies the first onto the second. It is an error if the last argument is not a directory and more than two non-option arguments are given.

Generally, files are written just as they are read. For exceptions, see the '`--sparse`' option below.

By default, `cp` does not copy directories. However, the '`-R`', '`-a`', and '`-r`' options cause `cp` to copy recursively by descending into source directories and copying files to corresponding destination directories.

By default, `cp` follows symbolic links only when not copying recursively. This default can be overridden with the '`--archive`' ('`-a`'), '`-d`', '`--dereference`' ('`-L`'), '`--no-dereference`' ('`-P`'), and '`-H`' options. If more than one of these options is specified, the last one silently overrides the others.

By default, `cp` copies the contents of special files only when not copying recursively. This default can be overridden with the '`--copy-contents`' option.

`cp` generally refuses to copy a file onto itself, with the following exception: if '`--force --backup`' is specified with *source* and *dest* identical, and referring to a regular file, `cp` will make a backup file, either regular or numbered, as specified in the usual ways (see Section 2.1 [Backup options], page 2). This is useful when you simply want to make a backup of an existing file before changing it.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-a`'
'`--archive`'
: Preserve as much as possible of the structure and attributes of the original files in the copy (but do not attempt to preserve internal directory structure; i.e., '`ls -U`' may list the entries in a copied directory in a different order). Equivalent to '`-dpPR`'.

'`-b`'
'`--backup[=method]`'
: See Section 2.1 [Backup options], page 2. Make a backup of each file that would otherwise be overwritten or removed. As a special case, `cp` makes a

backup of *source* when the force and backup options are given and *source* and *dest* are the same name for an existing, regular file. One useful application of this combination of options is this tiny Bourne shell script:

```
#!/bin/sh
# Usage: backup FILE...
# Create a GNU-style backup of each listed FILE.
for i; do
  cp --backup --force "$i" "$i"
done
```

'`--copy-contents`'
: If copying recursively, copy the contents of any special files (e.g., FIFOs and device files) as if they were regular files. This means trying to read the data in each source file and writing it to the destination. It is usually a mistake to use this option, as it normally has undesirable effects on special files like FIFOs and the ones typically found in the '`/dev`' directory. In most cases, `cp -R --copy-contents` will hang indefinitely trying to read from FIFOs and special files like '`/dev/console`', and it will fill up your destination disk if you use it to copy '`/dev/zero`'. This option has no effect unless copying recursively, and it does not affect the copying of symbolic links.

'`-d`'
: Copy symbolic links as symbolic links rather than copying the files that they point to, and preserve hard links between source files in the copies. Equivalent to '`--no-dereference --preserve=links`'.

'`-f`'
'`--force`'
: When copying without this option and an existing destination file cannot be opened for writing, the copy fails. However, with '`--force`'), when a destination file cannot be opened, `cp` then unlinks it and tries to open it again. Contrast this behavior with that enabled by '`--link`' and '`--symbolic-link`', whereby the destination file is never opened but rather is unlinked unconditionally. Also see the description of '`--remove-destination`'.

'`-H`'
: If a command line argument specifies a symbolic link, then copy the file it points to rather than the symbolic link itself. However, copy (preserving its nature) any symbolic link that is encountered via recursive traversal.

'`-i`'
'`--interactive`'
: Prompt whether to overwrite existing regular destination files.

'`-l`'
'`--link`'
: Make hard links instead of copies of non-directories.

'`-L`'
'`--dereference`'
: Always follow symbolic links.

'`-P`'
'`--no-dereference`'
: Copy symbolic links as symbolic links rather than copying the files that they point to.

'-p'
'--*preserve*[=*attribute_list*]'

Preserve the specified attributes of the original files. If specified, the *attribute_list* must be a comma-separated list of one or more of the following strings:

'mode' Preserve the permission attributes.

'ownership'
Preserve the owner and group. On most modern systems, only the super-user may change the owner of a file, and regular users may preserve the group ownership of a file only if they happen to be a member of the desired group.

'timestamps'
Preserve the times of last access and last modification.

'links' Preserve in the destination files any links between corresponding source files.

'all' Preserve all file attributes. Equivalent to specifying all of the above.

Using '--preserve' with no *attribute_list* is equivalent to '--preserve=mode,ownership,timestamps'.

In the absence of this option, each destination file is created with the permissions of the corresponding source file, minus the bits set in the umask and minus the set-user-id and set-group-id bits. See Chapter 26 [File permissions], page 135.

'--*no-preserve*=*attribute_list*'

Do not preserve the specified attributes. The *attribute_list* has the same form as for '--preserve'.

'--parents'

Form the name of each destination file by appending to the target directory a slash and the specified name of the source file. The last argument given to cp must be the name of an existing directory. For example, the command:

```
cp --parents a/b/c existing_dir
```

copies the file 'a/b/c' to 'existing_dir/a/b/c', creating any missing intermediate directories.

'--*reply*[=*how*]'

Using '--reply=yes' makes cp act as if 'yes' were given as a response to every prompt about a destination file. That effectively cancels any preceding '--interactive' or '-i' option. Specify '--reply=no' to make cp act as if 'no' were given as a response to every prompt about a destination file. Specify '--reply=query' to make cp prompt the user about each existing destination file.

'-R'
'-r'
'--recursive'

Copy directories recursively. Symbolic links are not followed by default; see the '--archive' ('-a'), '-d', '--dereference' ('-L'), '--no-dereference' ('-P'), and '-H' options. Special files are copied by creating a destination file of the same type as the source; see the '--copy-contents' option. It is not portable to use '-r' to copy symbolic links or special files. On some non-GNU systems, '-r' implies the equivalent of '-L' and '--copy-contents' for historical reasons. Also, it is not portable to use '-R' to copy symbolic links unless you also specify '-P', as POSIX allows implementations that dereference symbolic links by default.

'--remove-destination'
: Remove each existing destination file before attempting to open it (contrast with '-f' above).

'--sparse=*when*'
: A *sparse file* contains *holes*—a sequence of zero bytes that does not occupy any physical disk blocks; the 'read' system call reads these as zeroes. This can both save considerable disk space and increase speed, since many binary files contain lots of consecutive zero bytes. By default, `cp` detects holes in input source files via a crude heuristic and makes the corresponding output file sparse as well.

  The *when* value can be one of the following:

  'auto' The default behavior: the output file is sparse if the input file is sparse.

  'always' Always make the output file sparse. This is useful when the input file resides on a filesystem that does not support sparse files (the most notable example is 'efs' filesystems in SGI IRIX 5.3 and earlier), but the output file is on another type of filesystem.

  'never' Never make the output file sparse. This is useful in creating a file for use with the `mkswap` command, since such a file must not have any holes.

'--*strip-trailing-slashes*'
: Remove any trailing slashes from each *source* argument. See Section 2.4 [Trailing slashes], page 6.

'-s'
'--symbolic-link'
: Make symbolic links instead of copies of non-directories. All source file names must be absolute (starting with '/') unless the destination files are in the current directory. This option merely results in an error message on systems that do not support symbolic links.

'-S *suffix*'
'--suffix=*suffix*'
: Append *suffix* to each backup file made with '-b'. See Section 2.1 [Backup options], page 2.

'--*target-directory=directory*'
: Specify the destination *directory*. See Section 2.3 [Target directory], page 5.

'-v'
'--verbose'
: Print the name of each file before copying it.

'-V *method*'
'--version-control=*method*'
: Change the type of backups made with '-b'. The *method* argument can be 'none' (or 'off'), 'numbered' (or 't'), 'existing' (or 'nil'), or 'never' (or 'simple'). See Section 2.1 [Backup options], page 2.

'-x'
'--one-file-system'
: Skip subdirectories that are on different filesystems from the one that the copy started on. However, mount point directories *are* copied.

11.2 dd: Convert and copy a file

dd copies a file (from standard input to standard output, by default) with a changeable I/O block size, while optionally performing conversions on it. Synopsis:

```
dd [option]...
```

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

The numeric-valued options below (*bytes* and *blocks*) can be followed by a multiplier: 'b'=512, 'c'=1, 'w'=2, 'x*m*'=*m*, or any of the standard block size suffixes like 'k'=1024 (see Section 2.2 [Block size], page 3).

Use different dd invocations to use different block sizes for skipping and I/O. For example, the following shell commands copy data in 512 KiB blocks between a disk and a tape, but do not save or restore a 4 KiB label at the start of the disk:

```
disk=/dev/rdsk/c0t1d0s2
tape=/dev/rmt/0

# Copy all but the label from disk to tape.
(dd bs=4k skip=1 count=0 && dd bs=512k) <$disk >$tape

# Copy from tape back to disk, but leave the disk label alone.
(dd bs=4k seek=1 count=0 && dd bs=512k) <$tape >$disk
```

'if=*file*'
: Read from *file* instead of standard input.

'of=*file*'
: Write to *file* instead of standard output. Unless 'conv=notrunc' is given, dd truncates *file* to zero bytes (or the size specified with 'seek=').

'ibs=*bytes*'
: Read *bytes* bytes at a time.

'obs=*bytes*'
: Write *bytes* bytes at a time.

'bs=*bytes*'
: Both read and write *bytes* bytes at a time. This overrides 'ibs' and 'obs'.

'cbs=*bytes*'
: Convert *bytes* bytes at a time.

'skip=*blocks*'
: Skip *blocks* 'ibs'-byte blocks in the input file before copying.

'seek=*blocks*'
: Skip *blocks* 'obs'-byte blocks in the output file before copying.

'count=*blocks*'
: Copy *blocks* 'ibs'-byte blocks from the input file, instead of everything until the end of the file.

'conv=*conversion*[,*conversion*]...'
: Convert the file as specified by the *conversion* argument(s). (No spaces around any comma(s).)

  Conversions:

  'ascii' Convert EBCDIC to ASCII.

  'ebcdic' Convert ASCII to EBCDIC.

  'ibm' Convert ASCII to alternate EBCDIC.

  'block' For each line in the input, output 'cbs' bytes, replacing the input newline with a space and padding with spaces as necessary.

  'unblock' Replace trailing spaces in each 'cbs'-sized input block with a newline.

  'lcase' Change uppercase letters to lowercase.

  'ucase' Change lowercase letters to uppercase.

  'swab' Swap every pair of input bytes. GNU `dd`, unlike others, works when an odd number of bytes are read—the last byte is simply copied (since there is nothing to swap it with).

  'noerror' Continue after read errors.

  'notrunc' Do not truncate the output file.

  'sync' Pad every input block to size of 'ibs' with trailing zero bytes. When used with 'block' or 'unblock', pad with spaces instead of zero bytes.

11.3 `install`: Copy files and set attributes

`install` copies files while setting their permission modes and, if possible, their owner and group. Synopses:

```
install [option]... source dest
install [option]... source... directory
install -d [option]... directory...
```

In the first of these, the *source* file is copied to the *dest* target file. In the second, each of the *source* files are copied to the destination *directory*. In the last, each *directory* (and any missing parent directories) is created.

`install` is similar to `cp`, but allows you to control the attributes of destination files. It is typically used in Makefiles to copy programs into their destination directories. It refuses to copy files onto themselves.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-b`'
'`--backup[=`*method*`]`'
: See Section 2.1 [Backup options], page 2. Make a backup of each file that would otherwise be overwritten or removed.

'`-c`'
: Ignored; for compatibility with old Unix versions of `install`.

'`-d`'
'`--directory`'
: Create each given directory and any missing parent directories, setting the owner, group and mode as given on the command line or to the defaults. It also gives any parent directories it creates those attributes. (This is different from the SunOS 4.x `install`, which gives directories that it creates the default attributes.)

'`-g` *group*'
'`--group=`*group*'
: Set the group ownership of installed files or directories to *group*. The default is the process' current group. *group* may be either a group name or a numeric group id.

'`-m` *mode*'
'`--mode=`*mode*'
: Set the permissions for the installed file or directory to *mode*, which can be either an octal number, or a symbolic mode as in `chmod`, with 0 as the point of departure (see Chapter 26 [File permissions], page 135). The default mode is '`u=rwx,go=rx`'—read, write, and execute for the owner, and read and execute for group and other.

'`-o` *owner*'
'`--owner=`*owner*'
: If `install` has appropriate privileges (is run as root), set the ownership of installed files or directories to *owner*. The default is `root`. *owner* may be either a user name or a numeric user ID.

'`-p`'
'`--preserve-timestamps`'
: Set the time of last access and the time of last modification of each installed file to match those of each corresponding original file. When a file is installed without this option, its last access and last modification times are both set to the time of installation. This option is useful if you want to use the last modification times of installed files to keep track of when they were last built as opposed to when they were last installed.

'`-s`'
'`--strip`'
: Strip the symbol tables from installed binary executables.

'-S *suffix*'
'--suffix=*suffix*'
: Append *suffix* to each backup file made with '-b'. See Section 2.1 [Backup options], page 2.

'--target-directory=*directory*'
: Specify the destination *directory*. See Section 2.3 [Target directory], page 5.

'-v'
'--verbose'
: Print the name of each file before copying it.

'-V *method*'
'--version-control=*method*'
: Change the type of backups made with '-b'. The *method* argument can be 'none' (or 'off'), 'numbered' (or 't'), 'existing' (or 'nil'), or 'never' (or 'simple'). See Section 2.1 [Backup options], page 2.

11.4 mv: Move (rename) files

mv moves or renames files (or directories). Synopsis:

```
mv [option]... source dest
mv [option]... source... directory
```

If the last argument names an existing directory, mv moves each other given file into a file with the same name in that directory. Otherwise, if only two files are given, it renames the first as the second. It is an error if the last argument is not a directory and more than two files are given.

mv can move any type of file from one filesystem to another. Prior to version 4.0 of the fileutils, mv could move only regular files between filesystems. For example, now mv can move an entire directory hierarchy including special device files from one partition to another. It first uses some of the same code that's used by cp -a to copy the requested directories and files, then (assuming the copy succeeded) it removes the originals. If the copy fails, then the part that was copied to the destination partition is removed. If you were to copy three directories from one partition to another and the copy of the first directory succeeded, but the second didn't, the first would be left on the destination partition and the second and third would be left on the original partition.

If a destination file exists but is normally unwritable, standard input is a terminal, and the '-f' or '--force' option is not given, mv prompts the user for whether to replace the file. (You might own the file, or have write permission on its directory.) If the response does not begin with 'y' or 'Y', the file is skipped.

Warning: If you try to move a symlink that points to a directory, and you specify the symlink with a trailing slash, then mv doesn't move the symlink but instead moves the directory referenced by the symlink. See Section 2.4 [Trailing slashes], page 6.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-b'
'--*backup*[=*method*]'
: See Section 2.1 [Backup options], page 2. Make a backup of each file that would otherwise be overwritten or removed.

'-f'
'--force'
: Do not prompt the user before removing a destination file.

'-i'
'--interactive'
: Prompt whether to overwrite each existing destination file, regardless of its permissions. If the response does not begin with 'y' or 'Y', the file is skipped.

'--*reply*[=*how*]'
: Specifying '--reply=yes' is equivalent to using '--force'. Specify '--reply=no' to make mv act as if 'no' were given as a response to every prompt about a destination file. Specify '--reply=query' to make mv prompt the user about each existing destination file.

'-u'
'--update'
: Do not move a non-directory that has an existing destination with the same or newer modification time.

'-v'
'--verbose'
: Print the name of each file before moving it.

'--*strip-trailing-slashes*'
: Remove any trailing slashes from each *source* argument. See Section 2.4 [Trailing slashes], page 6.

'-S *suffix*'
'--suffix=*suffix*'
: Append *suffix* to each backup file made with '-b'. See Section 2.1 [Backup options], page 2.

'--*target-directory*=*directory*'
: Specify the destination *directory*. See Section 2.3 [Target directory], page 5.

'-V *method*'
'--version-control=*method*'
: Change the type of backups made with '-b'. The *method* argument can be 'none' (or 'off'), 'numbered' (or 't'), 'existing' (or 'nil'), or 'never' (or 'simple'). See Section 2.1 [Backup options], page 2.

11.5 rm: Remove files or directories

rm removes each given *file*. By default, it does not remove directories. Synopsis:

```
rm [option]... [file]...
```

If a file is unwritable, standard input is a terminal, and the '-f' or '--force' option is not given, or the '-i' or '--interactive' option *is* given, rm prompts the user for whether to remove the file. If the response does not begin with 'y' or 'Y', the file is skipped.

Warning: If you use `rm` to remove a file, it is usually possible to recover the contents of that file. If you want more assurance that the contents are truly unrecoverable, consider using `shred`.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-d'
'--directory'
: Attempt to remove directories using the `unlink` function rather than the `rmdir` function, and don't require a directory to be empty before trying to unlink it. This works only if you have appropriate privileges and if your operating system supports `unlink` for directories. Because unlinking a directory causes any files in the deleted directory to become unreferenced, it is wise to `fsck` the filesystem after doing this.

'-f'
'--force'
: Ignore nonexistent files and never prompt the user. Ignore any previous '--interactive' ('-i') option.

'-i'
'--interactive'
: Prompt whether to remove each file. If the response does not begin with 'y' or 'Y', the file is skipped. Ignore any previous '--force' ('-f') option.

'-r'
'-R'
'--recursive'
: Remove the contents of directories recursively.

'-v'
'--verbose'
: Print the name of each file before removing it.

One common question is how to remove files whose names begin with a '-'. GNU `rm`, like every program that uses the `getopt` function to parse its arguments, lets you use the '--' option to indicate that all following arguments are non-options. To remove a file called '-f' in the current directory, you could type either:

```
rm -- -f
```

or:

```
rm ./-f
```

The Unix `rm` program's use of a single '-' for this purpose predates the development of the getopt standard syntax.

11.6 `shred`: Remove files more securely

`shred` overwrites devices or files, to help prevent even very expensive hardware from recovering the data.

Ordinarily when you remove a file (see Section 11.5 [rm invocation], page 76), the data is not actually destroyed. Only the index listing where the file is stored is destroyed, and the storage is made available for reuse. There are undelete utilities that will attempt to reconstruct the index and can bring the file back if the parts were not reused.

On a busy system with a nearly-full drive, space can get reused in a few seconds. But there is no way to know for sure. If you have sensitive data, you may want to be sure that recovery is not possible by actually overwriting the file with non-sensitive data.

However, even after doing that, it is possible to take the disk back to a laboratory and use a lot of sensitive (and expensive) equipment to look for the faint "echoes" of the original data underneath the overwritten data. If the data has only been overwritten once, it's not even that hard.

The best way to remove something irretrievably is to destroy the media it's on with acid, melt it down, or the like. For cheap removable media like floppy disks, this is the preferred method. However, hard drives are expensive and hard to melt, so the `shred` utility tries to achieve a similar effect non-destructively.

This uses many overwrite passes, with the data patterns chosen to maximize the damage they do to the old data. While this will work on floppies, the patterns are designed for best effect on hard drives. For more details, see the source code and Peter Gutmann's paper *Secure Deletion of Data from Magnetic and Solid-State Memory*, from the proceedings of the Sixth USENIX Security Symposium (San Jose, California, 22–25 July, 1996). The paper is also available online `http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html`.

Please note that `shred` relies on a very important assumption: that the filesystem overwrites data in place. This is the traditional way to do things, but many modern filesystem designs do not satisfy this assumption. Exceptions include:

- Log-structured or journaled filesystems, such as those supplied with AIX and Solaris, and JFS, ReiserFS, XFS, Ext3, etc.
- Filesystems that write redundant data and carry on even if some writes fail, such as RAID-based filesystems.
- Filesystems that make snapshots, such as Network Appliance's NFS server.
- Filesystems that cache in temporary locations, such as NFS version 3 clients.
- Compressed filesystems.

If you are not sure how your filesystem operates, then you should assume that it does not overwrite data in place, which means that shred cannot reliably operate on regular files in your filesystem.

Generally speaking, it is more reliable to shred a device than a file, since this bypasses the problem of filesystem design mentioned above. However, even shredding devices is not always completely reliable. For example, most disks map out bad sectors invisibly to the application; if the bad sectors contain sensitive data, `shred` won't be able to destroy it.

`shred` makes no attempt to detect or report this problem, just as it makes no attempt to do anything about backups. However, since it is more reliable to shred devices than files, `shred` by default does not truncate or remove the output file. This default is more suitable for devices, which typically cannot be truncated and should not be removed.

Finally, consider the risk of backups and mirrors. File system backups and remote mirrors may contain copies of the file that cannot be removed, and that will allow a shredded file to be recovered later. So if you keep any data you may later want to destroy using `shred`, be sure that it is not backed up or mirrored.

```
shred [option]... file[...]
```

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-f'
'--force' Override file permissions if necessary to allow overwriting.

'-*NUMBER*'
'-n *NUMBER*'
'--iterations=*NUMBER*'
: By default, `shred` uses 25 passes of overwrite. This is enough for all of the useful overwrite patterns to be used at least once. You can reduce this to save time, or increase it if you have a lot of time to waste.

'-s *BYTES*'
'--size=*BYTES*'
: Shred the first *BYTES* bytes of the file. The default is to shred the whole file. *BYTES* can be followed by a size specification like 'K', 'M', or 'G' to specify a multiple. See Section 2.2 [Block size], page 3.

'-u'
'--remove'
: After shredding a file, truncate it (if possible) and then remove it. If a file has multiple links, only the named links will be removed.

'-v'
'--verbose'
: Display status updates as sterilization proceeds.

'-x'
'--exact' Normally, shred rounds the file size up to the next multiple of the filesystem block size to fully erase the last block of the file. This option suppresses that behavior. Thus, by default if you shred a 10-byte file on a system with 512-byte blocks, the resulting file will be 512 bytes long. With this option, shred does not increase the size of the file.

'-z'
'--zero' Normally, the last pass that `shred` writes is made up of random data. If this would be conspicuous on your hard drive (for example, because it looks like encrypted data), or you just think it's tidier, the '`--zero`' option adds an additional overwrite pass with all zero bits. This is in addition to the number of passes specified by the '`--iterations`' option.

'-' Shred standard output.

This argument is considered an option. If the common '`--`' option has been used to indicate the end of options on the command line, then '`-`' will be interpreted as an ordinary file name.

The intended use of this is to shred a removed temporary file. For example

```
i=`tempfile -m 0600`
exec 3<>"$i"
```

```
rm -- "$i"
echo "Hello, world" >&3
shred - >&3
exec 3>-
```

Note that the shell command '`shred - >file`' does not shred the contents of *file*, since it truncates *file* before invoking `shred`. Use the command '`shred file`' or (if using a Bourne-compatible shell) the command '`shred - 1<>file`' instead.

You might use the following command to erase all trace of the filesystem you'd created on the floppy disk in your first drive. That command takes about 20 minutes to erase a "1.44MB" (actually 1440 KiB) floppy.

```
shred --verbose /dev/fd0
```

Similarly, to erase all data on a selected partition of your hard disk, you could give a command like this:

```
shred --verbose /dev/sda5
```

12 Special file types

This chapter describes commands which create special types of files (and `rmdir`, which removes directories, one special file type).

Although Unix-like operating systems have markedly fewer special file types than others, not *everything* can be treated only as the undifferentiated byte stream of *normal files*. For example, when a file is created or removed, the system must record this information, which it does in a *directory*—a special type of file. Although you can read directories as normal files, if you're curious, in order for the system to do its job it must impose a structure, a certain order, on the bytes of the file. Thus it is a "special" type of file.

Besides directories, other special file types include named pipes (FIFOs), symbolic links, sockets, and so-called *special files*.

12.1 `link`: Make a hard link via the link syscall

`link` creates a single hard link at a time. It is a minimalist interface to the system-provided `link` function. See section "Hard Links" in *The GNU C Library Reference Manual*. Synopsis:

```
link filename linkname
```

filename must specify an existing file, and *linkname* must specify a nonexistent entry in an existing directory. `link` simply calls `link (filename, linkname)` to create the link.

12.2 `ln`: Make links between files

`ln` makes links between files. By default, it makes hard links; with the '`-s`' option, it makes symbolic (or *soft*) links. Synopses:

```
ln [option]... target [linkname]
ln [option]... target... directory
```

- If the last argument names an existing directory, `ln` creates a link to each *target* file in that directory, using the *targets*' names. (But see the description of the '`--no-dereference`' option below.)
- If two filenames are given, `ln` creates a link from the second to the first.
- If one *target* is given, `ln` creates a link to that file in the current directory.
- It is an error if the last argument is not a directory and more than two files are given. Without '`-f`' or '`-i`' (see below), `ln` will not remove an existing file. Use the '`--backup`' option to make `ln` rename existing files.

A *hard link* is another name for an existing file; the link and the original are indistinguishable. Technically speaking, they share the same inode, and the inode contains all the information about a file—indeed, it is not incorrect to say that the inode *is* the file. On all existing implementations, you cannot make a hard link to a directory, and hard links cannot cross filesystem boundaries. (These restrictions are not mandated by POSIX, however.)

Symbolic links (*symlinks* for short), on the other hand, are a special file type (which not all kernels support: System V release 3 (and older) systems lack symlinks) in which the link file actually refers to a different file, by name. When most operations (opening, reading, writing, and so on) are passed the symbolic link file, the kernel automatically *dereferences* the link and operates on the target of the link. But some operations (e.g., removing) work on the link file itself, rather than on its target. See section "Symbolic Links" in *The GNU C Library Reference Manual*.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-b`'
'`--`*`backup`*`[=`*`method`*`]`'
: See Section 2.1 [Backup options], page 2. Make a backup of each file that would otherwise be overwritten or removed.

'`-d`'
'`-F`'
'`--directory`'
: Allow the super-user to make hard links to directories.

'`-f`'
'`--force`'
: Remove existing destination files.

'`-i`'
'`--interactive`'
: Prompt whether to remove existing destination files.

'`-n`'
'`--no-dereference`'
: When given an explicit destination that is a symlink to a directory, treat that destination as if it were a normal file.

  When the destination is an actual directory (not a symlink to one), there is no ambiguity. The link is created in that directory. But when the specified destination is a symlink to a directory, there are two ways to treat the user's request. `ln` can treat the destination just as it would a normal directory and create the link in it. On the other hand, the destination can be viewed as a non-directory—as the symlink itself. In that case, `ln` must delete or backup that symlink before creating the new link. The default is to treat a destination that is a symlink to a directory just like a directory.

'`-s`'
'`--symbolic`'
: Make symbolic links instead of hard links. This option merely produces an error message on systems that do not support symbolic links.

'`-S` *`suffix`*'
'`--suffix=`*`suffix`*'
: Append *suffix* to each backup file made with '`-b`'. See Section 2.1 [Backup options], page 2.

'`--`*`target-directory`*`=`*`directory`*'
: Specify the destination *directory*. See Section 2.3 [Target directory], page 5.

'-v'
'--verbose'
: Print the name of each file before linking it.

'-V *method*'
'--version-control=*method*'
: Change the type of backups made with '-b'. The *method* argument can be 'none' (or 'off'), 'numbered' (or 't'), 'existing' (or 'nil'), or 'never' (or 'simple'). See Section 2.1 [Backup options], page 2.

Examples:

```
ln -s /some/name  # creates link ./name pointing to /some/name
ln -s /some/name myname  # creates link ./myname pointing to /some/name
ln -s a b ..      # creates links ../a and ../b pointing to ./a and ./b
```

12.3 `mkdir`: Make directories

`mkdir` creates directories with the specified names. Synopsis:

```
mkdir [option]... name...
```

If a *name* is an existing file but not a directory, `mkdir` prints a warning message on stderr and will exit with a status of 1 after processing any remaining *name*s. The same is done when a *name* is an existing directory and the -p option is not given. If a *name* is an existing directory and the -p option is given, `mkdir` will ignore it. That is, `mkdir` will not print a warning, raise an error, or change the mode of the directory (even if the -m option is given), and will move on to processing any remaining *name*s.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-m *mode*'
'--mode=*mode*'
: Set the mode of created directories to *mode*, which is symbolic as in `chmod` and uses 'a=rwx' (read, write and execute allowed for everyone) minus the bits set in the umask for the point of the departure. See Chapter 26 [File permissions], page 135.

'-p'
'--parents'
: Make any missing parent directories for each argument. The mode for parent directories is set to the umask modified by 'u+wx'. Ignore arguments corresponding to existing directories.

'-v'

'--verbose'
: Print a message for each created directory. This is most useful with '--parents'.

12.4 `mkfifo`: Make FIFOs (named pipes)

`mkfifo` creates FIFOs (also called *named pipes*) with the specified names. Synopsis:

```
mkfifo [option] name...
```

A *FIFO* is a special file type that permits independent processes to communicate. One process opens the FIFO file for writing, and another for reading, after which data can flow as with the usual anonymous pipe in shells or elsewhere.

The program accepts the following option. Also see Chapter 2 [Common options], page 2.

'`-m` *mode*'
'`--mode=`*mode*'
: Set the mode of created FIFOs to *mode*, which is symbolic as in `chmod` and uses '`a=rw`' (read and write allowed for everyone) minus the bits set in the umask for the point of departure. See Chapter 26 [File permissions], page 135.

12.5 `mknod`: Make block or character special files

`mknod` creates a FIFO, character special file, or block special file with the specified name. Synopsis:

```
mknod [option]... name type [major minor]
```

Unlike the phrase "special file type" above, the term *special file* has a technical meaning on Unix: something that can generate or receive data. Usually this corresponds to a physical piece of hardware, e.g., a printer or a disk. (These files are typically created at system-configuration time.) The `mknod` command is what creates files of this type. Such devices can be read either a character at a time or a "block" (many characters) at a time, hence we say there are *block special* files and *character special* files.

The arguments after *name* specify the type of file to make:

'`p`' for a FIFO

'`b`' for a block special file

'`c`' for a character special file

When making a block or character special file, the major and minor device numbers must be given after the file type.

The program accepts the following option. Also see Chapter 2 [Common options], page 2.

'`-m` *mode*'
'`--mode=`*mode*'
: Set the mode of created files to *mode*, which is symbolic as in `chmod` and uses '`a=rw`' minus the bits set in the umask as the point of departure. See Chapter 26 [File permissions], page 135.

12.6 `rmdir`: Remove empty directories

`rmdir` removes empty directories. Synopsis:

```
rmdir [option]... directory...
```

If any *directory* argument does not refer to an existing empty directory, it is an error.

The program accepts the following option. Also see Chapter 2 [Common options], page 2.

'`--ignore-fail-on-non-empty`'
: Ignore each failure to remove a directory that is solely because the directory is non-empty.

'`-p`'
'`--parents`'
: Remove *directory*, then try to remove each component of *directory*. So, for example, '`rmdir -p a/b/c`' is similar to '`rmdir a/b/c a/b a`'. As such, it fails if any of those directories turns out not to be empty. Use the '`--ignore-fail-on-non-empty`' option to make it so such a failure does not evoke a diagnostic and does not cause `rmdir` to exit unsuccessfully.

'`-v`'

'`--verbose`'
: Give a diagnostic for each successful removal. *directory* is removed.

See Section 11.5 [rm invocation], page 76, for how to remove non-empty directories (recursively).

12.7 `unlink`: Remove files via the unlink syscall

`unlink` deletes a single specified file name. It is a minimalist interface to the system-provided `unlink` function. See section "Deleting Files" in *The GNU C Library Reference Manual*. Synopsis:

```
unlink filename
```

On some systems `unlink` can be used to delete the name of a directory. On others, it can be used that way only by a privileged user. In the GNU system `unlink` can never delete the name of a directory.

By default, `unlink` honors the '`--help`' and '`--version`' options. That makes it a little harder to remove files named `--help` and `--version`, so when the environment variable `POSIXLY_CORRECT` is set, `unlink` treats such a command line arguments not as an option, but as an operand.

13 Changing file attributes

A file is not merely its contents, a name, and a file type (see Chapter 12 [Special file types], page 81). A file also has an owner (a userid), a group (a group id), permissions (what the owner can do with the file, what people in the group can do, and what everyone else can do), various timestamps, and other information. Collectively, we call these a file's *attributes*.

These commands change file attributes.

13.1 `chown`: Change file owner and group

`chown` changes the user and/or group ownership of each given *file* to *new-owner* or to the user and group of an existing reference file. Synopsis:

```
chown [option]... {new-owner | --reference=ref_file} file...
```

If used, *new-owner* specifies the new owner and/or group as follows (with no embedded white space):

```
[owner] [ [:] [group] ]
```

Specifically:

owner If only an *owner* (a user name or numeric user id) is given, that user is made the owner of each given file, and the files' group is not changed.

owner':'*group*
If the *owner* is followed by a colon and a *group* (a group name or numeric group id), with no spaces between them, the group ownership of the files is changed as well (to *group*).

owner':' If a colon but no group name follows *owner*, that user is made the owner of the files and the group of the files is changed to *owner*'s login group.

':'*group* If the colon and following *group* are given, but the owner is omitted, only the group of the files is changed; in this case, `chown` performs the same function as `chgrp`.

You may use '`.`' in place of the '`:`' separator. This is a GNU extension for compatibility with older scripts. New scripts should avoid the use of '`.`' because GNU `chown` may fail if *owner* contains '`.`' characters.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-c`'
'`--changes`'
Verbosely describe the action for each *file* whose ownership actually changes.

'`-f`'
'`--silent`'
'`--quiet`' Do not print error messages about files whose ownership cannot be changed.

'`--from=old-owner`'
Change a *file*'s ownership only if it has current attributes specified by *old-owner*. *old-owner* has the same form as *new-owner* described above. This option is

useful primarily from a security standpoint in that it narrows considerably the window of potential abuse. For example, to reflect a UID numbering change for one user's files without an option like this, `root` might run

```
find / -owner OLDUSER -print0 | xargs -0 chown NEWUSER
```

But that is dangerous because the interval between when the `find` tests the existing file's owner and when the `chown` is actually run may be quite large. One way to narrow the gap would be to invoke chown for each file as it is found:

```
find / -owner OLDUSER -exec chown NEWUSER {} \;
```

But that is very slow if there are many affected files. With this option, it is safer (the gap is narrower still) though still not perfect:

```
chown -R --from=OLDUSER NEWUSER /
```

'`--dereference`'
Do not act on symbolic links themselves but rather on what they point to.

'`-h`'
'`--no-dereference`'
Act on symbolic links themselves instead of what they point to. This is the default. This mode relies on the `lchown` system call. On systems that do not provide the `lchown` system call, `chown` fails when a file specified on the command line is a symbolic link. By default, no diagnostic is issued for symbolic links encountered during a recursive traversal, but see '`--verbose`'.

'`--reference=`*`ref_file`*'
Change the user and group of each *file* to be the same as those of *ref_file*. If *ref_file* is a symbolic link, do not use the user and group of the symbolic link, but rather those of the file it refers to.

'`-v`'
'`--verbose`'
Output a diagnostic for every file processed. If a symbolic link is encountered during a recursive traversal on a system without the `lchown` system call, and '`--no-dereference`' is in effect, then issue a diagnostic saying neither the symbolic link nor its referent is being changed.

'`-R`'
'`--recursive`'
Recursively change ownership of directories and their contents.

13.2 `chgrp`: Change group ownership

`chgrp` changes the group ownership of each given *file* to *group* (which can be either a group name or a numeric group id) or to the group of an existing reference file. Synopsis:

```
chgrp [option]... {group | --reference=ref_file} file...
```

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-c'
'--changes'
: Verbosely describe the action for each *file* whose group actually changes.

'-f'
'--silent'
'--quiet'
: Do not print error messages about files whose group cannot be changed.

'--dereference'
: Do not act on symbolic links themselves but rather on what they point to.

'-h'
'--no-dereference'
: Act on symbolic links themselves instead of what they point to. This is the default. This mode relies on the `lchown` system call. On systems that do not provide the `lchown` system call, `chgrp` fails when a file specified on the command line is a symbolic link. By default, no diagnostic is issued for symbolic links encountered during a recursive traversal, but see '--verbose'.

'--reference=*ref_file*'
: Change the group of each *file* to be the same as that of *ref_file*. If *ref_file* is a symbolic link, do not use the group of the symbolic link, but rather that of the file it refers to.

'-v'
'--verbose'
: Output a diagnostic for every file processed. If a symbolic link is encountered during a recursive traversal on a system without the `lchown` system call, and '--no-dereference' is in effect, then issue a diagnostic saying neither the symbolic link nor its referent is being changed.

'-R'
'--recursive'
: Recursively change the group ownership of directories and their contents.

13.3 `chmod`: Change access permissions

`chmod` changes the access permissions of the named files. Synopsis:

`chmod [option]... {mode | --reference=ref_file} file...`

`chmod` never changes the permissions of symbolic links, since the `chmod` system call cannot change their permissions. This is not a problem since the permissions of symbolic links are never used. However, for each symbolic link listed on the command line, `chmod` changes the permissions of the pointed-to file. In contrast, `chmod` ignores symbolic links encountered during recursive directory traversals.

If used, *mode* specifies the new permissions. For details, see the section on Chapter 26 [File permissions], page 135.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-c'
'--changes'
: Verbosely describe the action for each *file* whose permissions actually changes.

'-f'
'--silent'
'--quiet' Do not print error messages about files whose permissions cannot be changed.

'-v'
'--verbose'
: Verbosely describe the action or non-action taken for every *file*.

'--reference=*ref_file*'
: Change the mode of each *file* to be the same as that of *ref_file*. See Chapter 26 [File permissions], page 135. If *ref_file* is a symbolic link, do not use the mode of the symbolic link, but rather that of the file it refers to.

'-R'
'--recursive'
: Recursively change permissions of directories and their contents.

13.4 `touch`: Change file timestamps

`touch` changes the access and/or modification times of the specified files. Synopsis:

```
touch [option]... file...
```

On older systems, `touch` supports an obsolete syntax, as follows. If the first *file* would be a valid argument to the '-t' option and no timestamp is given with any of the '-d', '-r', or '-t' options and the '--' argument is not given, that argument is interpreted as the time for the other files instead of as a file name. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '-t' instead.

Any *file* that does not exist is created empty.

If changing both the access and modification times to the current time, `touch` can change the timestamps for files that the user running it does not own but has write permission for. Otherwise, the user must own the files.

Although `touch` provides options for changing two of the times – the times of last access and modification – of a file, there is actually a third one as well: the inode change time. This is often referred to as a file's `ctime`. The inode change time represents the time when the file's meta-information last changed. One common example of this is when the permissions of a file change. Changing the permissions doesn't access the file, so the atime doesn't change, nor does it modify the file, so the mtime doesn't change. Yet, something about the file itself has changed, and this must be noted somewhere. This is the job of the ctime field. This is necessary, so that, for example, a backup program can make a fresh copy of the file, including the new permissions value. Another operation that modifies a file's ctime without affecting the others is renaming. In any case, it is not possible, in normal operations, for a user to change the ctime field to a user-specified value.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-a'
'--time=atime'
'--time=access'
'--time=use'
: Change the access time only.

'-c'
'--no-create'
: Do not create files that do not exist.

'-d'
'--date=time'
: Use *time* instead of the current time. It can contain month names, time zones, 'am' and 'pm', etc. See Chapter 27 [Date input formats], page 141.

'-f' Ignored; for compatibility with BSD versions of `touch`.

'-m'
'--time=mtime'
'--time=modify'
: Change the modification time only.

'-r *file*'
'--reference=*file*'
: Use the times of the reference *file* instead of the current time.

'-t [[CC]YY]MMDDhhmm[.ss]'
: Use the argument (optional four-digit or two-digit years, months, days, hours, minutes, optional seconds) instead of the current time. If the year is specified with only two digits, then *CC* is 20 for years in the range 0 ... 68, and 19 for years in 69 ... 99. If no digits of the year are specified, the argument is interpreted as a date in the current year.

14 Disk usage

No disk can hold an infinite amount of data. These commands report on how much disk storage is in use or available. (This has nothing much to do with how much *main memory*, i.e., RAM, a program is using when it runs; for that, you want `ps` or `pstat` or `swap` or some such command.)

14.1 `df`: Report filesystem disk space usage

`df` reports the amount of disk space used and available on filesystems. Synopsis:

```
df [option]... [file]...
```

With no arguments, `df` reports the space used and available on all currently mounted filesystems (of all types). Otherwise, `df` reports on the filesystem containing each argument *file*.

Normally the disk space is printed in units of 1024 bytes, but this can be overridden (see Section 2.2 [Block size], page 3). Non-integer quantities are rounded up to the next higher unit.

If an argument *file* is a disk device file containing a mounted filesystem, `df` shows the space available on that filesystem rather than on the filesystem containing the device node (i.e., the root filesystem). GNU `df` does not attempt to determine the disk usage on unmounted filesystems, because on most kinds of systems doing so requires extremely non-portable intimate knowledge of filesystem structures.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

‘`-a`’
‘`--all`’
: Include in the listing filesystems that have a size of 0 blocks, which are omitted by default. Such filesystems are typically special-purpose pseudo-filesystems, such as automounter entries. Also, filesystems of type “ignore” or “auto”, supported by some operating systems, are only included if this option is specified.

‘`-B` *size*’
‘`--block-size=`*size*’
: Scale sizes by *size* before printing them (see Section 2.2 [Block size], page 3). For example, ‘`-BG`’ prints sizes in units of 1,073,741,824 bytes.

‘`-h`’
‘`--human-readable`’
: Append a size letter to each size, such as ‘`M`’ for mebibytes. Powers of 1024 are used, not 1000; ‘`M`’ stands for 1,048,576 bytes. Use the ‘`-H`’ or ‘`--si`’ option if you prefer powers of 1000.

‘`-H`’
‘`--si`’
: Append an SI-style abbreviation to each size, such as ‘`MB`’ for megabytes. Powers of 1000 are used, not 1024; ‘`MB`’ stands for 1,000,000 bytes. Use the ‘`-h`’ or ‘`--human-readable`’ option if you prefer powers of 1024.

'-i'
'--inodes'
List inode usage information instead of block usage. An inode (short for index node) contains information about a file such as its owner, permissions, timestamps, and location on the disk.

'-k' Print sizes in 1024-byte blocks, overriding the default block size (see Section 2.2 [Block size], page 3). This option is equivalent to '--block-size=1K'.

'-l'
'--local' Limit the listing to local filesystems. By default, remote filesystems are also listed.

'--no-sync'
Do not invoke the `sync` system call before getting any usage data. This may make `df` run significantly faster on systems with many disks, but on some systems (notably SunOS) the results may be slightly out of date. This is the default.

'-P'
'--portability'
Use the POSIX output format. This is like the default format except for the following:

1. The information about each filesystem is always printed on exactly one line; a mount device is never put on a line by itself. This means that if the mount device name is more than 20 characters long (e.g., for some network mounts), the columns are misaligned.
2. The labels in the header output line are changed to conform to POSIX.

'--sync' Invoke the `sync` system call before getting any usage data. On some systems (notably SunOS), doing this yields more up to date results, but in general this option makes `df` much slower, especially when there are many or very busy filesystems.

'-t *fstype*'
'--type=*fstype*'
Limit the listing to filesystems of type *fstype*. Multiple filesystem types can be specified by giving multiple '-t' options. By default, nothing is omitted.

'-T'
'--print-type'
Print each filesystem's type. The types printed here are the same ones you can include or exclude with '-t' and '-x'. The particular types printed are whatever is supported by the system. Here are some of the common names (this list is certainly not exhaustive):

'nfs' An NFS filesystem, i.e., one mounted over a network from another machine. This is the one type name which seems to be used uniformly by all systems.

'4.2, ufs, efs...'
: A filesystem on a locally-mounted hard disk. (The system might even support more than one type here; Linux does.)

'hsfs, cdfs'
: A filesystem on a CD-ROM drive. HP-UX uses 'cdfs', most other systems use 'hsfs' ('hs' for "High Sierra").

'pcfs' An MS-DOS filesystem, usually on a diskette.

'-x *fstype*'
'--exclude-type=*fstype*'
: Limit the listing to filesystems not of type *fstype*. Multiple filesystem types can be eliminated by giving multiple '-x' options. By default, no filesystem types are omitted.

'-v' Ignored; for compatibility with System V versions of `df`.

14.2 `du`: Estimate file space usage

`du` reports the amount of disk space used by the specified files and for each subdirectory (of directory arguments). Synopsis:

```
du [option]... [file]...
```

With no arguments, `du` reports the disk space for the current directory. Normally the disk space is printed in units of 1024 bytes, but this can be overridden (see Section 2.2 [Block size], page 3). Non-integer quantities are rounded up to the next higher unit.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-a'
'--all' Show counts for all files, not just directories.

'-b'
'--bytes' Print sizes in bytes, overriding the default block size (see Section 2.2 [Block size], page 3).

'-B *size*'
'--block-size=*size*'
: Scale sizes by *size* before printing them (see Section 2.2 [Block size], page 3). For example, '-BG' prints sizes in units of 1,073,741,824 bytes.

'-c'
'--total' Print a grand total of all arguments after all arguments have been processed. This can be used to find out the total disk usage of a given set of files or directories.

'-D'
'--dereference-args'
: Dereference symbolic links that are command line arguments. Does not affect other symbolic links. This is helpful for finding out the disk usage of directories, such as '/usr/tmp', which are often symbolic links.

'-h'
'--human-readable'
: Append a size letter to each size, such as 'M' for mebibytes. Powers of 1024 are used, not 1000; 'M' stands for 1,048,576 bytes. Use the '-H' or '--si' option if you prefer powers of 1000.

'-H'
'--si'
: Append an SI-style abbreviation to each size, such as 'MB' for megabytes. Powers of 1000 are used, not 1024; 'MB' stands for 1,000,000 bytes. Use the '-h' or '--human-readable' option if you prefer powers of 1024.

'-k'
: Print sizes in 1024-byte blocks, overriding the default block size (see Section 2.2 [Block size], page 3). This option is equivalent to '--block-size=1K'.

'-l'
'--count-links'
: Count the size of all files, even if they have appeared already (as a hard link).

'-L'
'--dereference'
: Dereference symbolic links (show the disk space used by the file or directory that the link points to instead of the space used by the link).

'--max-depth=*DEPTH*'
: Show the total for each directory (and file if –all) that is at most MAX_DEPTH levels down from the root of the hierarchy. The root is at level 0, so `du --max-depth=0` is equivalent to `du -s`.

'-s'
'--summarize'
: Display only a total for each argument.

'-S'
'--separate-dirs'
: Report the size of each directory separately, not including the sizes of subdirectories.

'-x'
'--one-file-system'
: Skip directories that are on different filesystems from the one that the argument being processed is on.

'--exclude=*PATTERN*'
: When recursing, skip subdirectories or files matching *PATTERN*. For example, `du --exclude='*.o'` excludes files whose names end in '.o'.

'-X *FILE*'
'--exclude-from=*FILE*'
: Like '--exclude', except take the patterns to exclude from *FILE*, one per line. If *FILE* is '-', take the patterns from standard input.

On BSD systems, `du` reports sizes that are half the correct values for files that are NFS-mounted from HP-UX systems. On HP-UX systems, it reports sizes that are twice the correct values for files that are NFS-mounted from BSD systems. This is due to a flaw in HP-UX; it also affects the HP-UX `du` program.

14.3 `stat`: Report file or filesystem status

`stat` displays information about the specified file(s). Synopsis:

```
stat [option]... [file]...
```

With no option, `stat` reports all information about the given files. But it also can be used to report the information of the filesystems the given files are located on. If the files are links, `stat` can also give information about the files the links point to.

'`-f`'
'`--filesystem`'
: Report information about the filesystems where the given files are located instead of information about the files themselves.

'`-L`'
'`--dereference`'
: Change how `stat` treats symbolic links. With this option, `stat` acts on the file referenced by each symbolic link argument. Without it, `stat` acts on any symbolic link argument directly.

'`-t`'
'`--terse`'
: Print the information in terse form, suitable for parsing by other programs.

'`-c`'
'`--format`'
: Allow user to specify the output format.

Interpreted sequences for file stat are:

- %n - File name
- %N - Quoted File name with dereference if symbolic link
- %d - Device number in decimal
- %D - Device number in hex
- %i - Inode number
- %a - Access rights in octal
- %A - Access rights in human readable form
- %f - raw mode in hex
- %F - File type
- %h - Number of hard links
- %u - User Id of owner
- %U - User name of owner
- %g - Group Id of owner
- %G - Group name of owner
- %t - Major device type in hex

- %T - Minor device type in hex
- %s - Total size, in bytes
- %b - Number of blocks allocated
- %o - IO block size
- %x - Time of last access
- %X - Time of last access as seconds since Epoch
- %y - Time of last modification
- %Y - Time of last modification as seconds since Epoch
- %z - Time of last change
- %Z - Time of last change as seconds since Epoch

Interpreted sequences for filesystem stat are:

- %n - File name
- %i - File System id in hex
- %l - Maximum length of filenames
- %t - Type in hex
- %T - Type in human readable form
- %b - Total data blocks in file system
- %f - Free blocks in file system
- %a - Free blocks available to non-superuser
- %s - Optimal transfer block size
- %c - Total file nodes in file system

14.4 `sync`: Synchronize data on disk with memory

`sync` writes any data buffered in memory out to disk. This can include (but is not limited to) modified superblocks, modified inodes, and delayed reads and writes. This must be implemented by the kernel; The `sync` program does nothing but exercise the `sync` system call.

The kernel keeps data in memory to avoid doing (relatively slow) disk reads and writes. This improves performance, but if the computer crashes, data may be lost or the filesystem corrupted as a result. `sync` ensures everything in memory is written to disk.

Any arguments are ignored, except for a lone '`--help`' or '`--version`' (see Chapter 2 [Common options], page 2).

15 Printing text

This section describes commands that display text strings.

15.1 `echo`: Print a line of text

`echo` writes each given *string* to standard output, with a space between each and a newline after the last one. Synopsis:

```
echo [option]... [string]...
```

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-n`' Do not output the trailing newline.

'`-e`' Enable interpretation of the following backslash-escaped characters in each *string*:

- '`\a`' alert (bell)
- '`\b`' backspace
- '`\c`' suppress trailing newline
- '`\f`' form feed
- '`\n`' new line
- '`\r`' carriage return
- '`\t`' horizontal tab
- '`\v`' vertical tab
- '`\\`' backslash
- '`\`*nnn*' the character whose ASCII code is *nnn* (octal); if *nnn* is not a valid octal number, it is printed literally.

15.2 `printf`: Format and print data

`printf` does formatted printing of text. Synopsis:

```
printf format [argument]...
```

`printf` prints the *format* string, interpreting '`%`' directives and '`\`' escapes in the same way as the C `printf` function. The *format* argument is re-used as necessary to convert all of the given *arguments*.

`printf` has one additional directive, '`%b`', which prints its argument string with '`\`' escapes interpreted in the same way as in the *format* string.

`printf` interprets '`\0ooo`' in *format* as an octal number (if *ooo* is 0 to 3 octal digits) specifying a character to print, and '`\xhh`' as a hexadecimal number (if *hh* is 1 to 2 hex digits) specifying a character to print.

`printf` interprets two character syntaxes introduced in ISO C 99: '`\u`' for 16-bit Unicode characters, specified as 4 hex digits *hhhh*, and '`\U`' for 32-bit Unicode characters, specified as 8 hex digits *hhhhhhhh*. `printf` outputs the Unicode characters according to the LC_CTYPE part of the current locale, i.e. depending on the values of the environment variables `LC_ALL`, `LC_CTYPE`, `LANG`.

The processing of '`\u`' and '`\U`' requires a full-featured `iconv` facility. It is activated on systems with glibc 2.2 (or newer), or when `libiconv` is installed prior to this package. Otherwise the use of '`\u`' and '`\U`' will give an error message.

An additional escape, '`\c`', causes `printf` to produce no further output.

The only options are a lone '`--help`' or '`--version`'. See Chapter 2 [Common options], page 2.

The Unicode character syntaxes are useful for writing strings in a locale independent way. For example, a string containing the Euro currency symbol

```
$ /usr/local/bin/printf '\u20AC 14.95'
```

will be output correctly in all locales supporting the Euro symbol (ISO-8859-15, UTF-8, and others). Similarly, a Chinese string

```
$ /usr/local/bin/printf '\u4e2d\u6587'
```

will be output correctly in all Chinese locales (GB2312, BIG5, UTF-8, etc).

Note that in these examples, the full pathname of `printf` has been given, to distinguish it from the GNU `bash` builtin function `printf`.

For larger strings, you don't need to look up the hexadecimal code values of each character one by one. ASCII characters mixed with \u escape sequences is also known as the JAVA source file encoding. You can use GNU recode 3.5c (or newer) to convert strings to this encoding. Here is how to convert a piece of text into a shell script which will output this text in a locale-independent way:

```
$ LC_CTYPE=zh_CN.big5 /usr/local/bin/printf \
    '\u4e2d\u6587\n' > sample.txt
$ recode BIG5..JAVA < sample.txt \
    | sed -e "s|^|/usr/local/bin/printf '|" -e "s|$|\\\\n'|" \
    > sample.sh
```

15.3 yes: Print a string until interrupted

`yes` prints the command line arguments, separated by spaces and followed by a newline, forever until it is killed. If no arguments are given, it prints '`y`' followed by a newline forever until killed.

The only options are a lone '`--help`' or '`--version`'. See Chapter 2 [Common options], page 2.

16 Conditions

This section describes commands that are primarily useful for their exit status, rather than their output. Thus, they are often used as the condition of shell `if` statements, or as the last command in a pipeline.

16.1 `false`: Do nothing, unsuccessfully

`false` does nothing except return an exit status of 1, meaning *failure*. It can be used as a place holder in shell scripts where an unsuccessful command is needed.

By default, `false` honors the '`--help`' and '`--version`' options. However, that is contrary to POSIX, so when the environment variable `POSIXLY_CORRECT` is set, `false` ignores *all* command line arguments, including '`--help`' and '`--version`'.

This version of `false` is implemented as a C program, and is thus more secure and faster than a shell script implementation, and may safely be used as a dummy shell for the purpose of disabling accounts.

16.2 `true`: Do nothing, successfully

`true` does nothing except return an exit status of 0, meaning *success*. It can be used as a place holder in shell scripts where a successful command is needed, although the shell built-in command `:` (colon) may do the same thing faster. In most modern shells, `true` is a built-in command, so when you use '`true`' in a script, you're probably using the built-in command, not the one documented here.

By default, `true` honors the '`--help`' and '`--version`' options. However, that is contrary to POSIX, so when the environment variable `POSIXLY_CORRECT` is set, `true` ignores *all* command line arguments, including '`--help`' and '`--version`'.

This version of `true` is implemented as a C program, and is thus more secure and faster than a shell script implementation, and may safely be used as a dummy shell for the purpose of disabling accounts.

16.3 `test`: Check file types and compare values

`test` returns a status of 0 (true) or 1 (false) depending on the evaluation of the conditional expression *expr*. Each part of the expression must be a separate argument.

`test` has file status checks, string operators, and numeric comparison operators.

Because most shells have a built-in command by the same name, using the unadorned command name in a script or interactively may get you different functionality than that described here.

Besides the options below, `test` accepts a lone '`--help`' or '`--version`'. See Chapter 2 [Common options], page 2. A single non-option argument is also allowed: `test` returns true if the argument is not null.

16.3.1 File type tests

These options test for particular types of files. (Everything's a file, but not all files are the same!)

'-b *file*' True if *file* exists and is a block special device.

'-c *file*' True if *file* exists and is a character special device.

'-d *file*' True if *file* exists and is a directory.

'-f *file*' True if *file* exists and is a regular file.

'-h *file*'
'-L *file*' True if *file* exists and is a symbolic link.

'-p *file*' True if *file* exists and is a named pipe.

'-S *file*' True if *file* exists and is a socket.

'-t [*fd*]' True if *fd* is opened on a terminal. If *fd* is omitted, it defaults to 1 (standard output).

16.3.2 Access permission tests

These options test for particular access permissions.

'-g *file*' True if *file* exists and has its set-group-id bit set.

'-k *file*' True if *file* has its *sticky* bit set.

'-r *file*' True if *file* exists and is readable.

'-u *file*' True if *file* exists and has its set-user-id bit set.

'-w *file*' True if *file* exists and is writable.

'-x *file*' True if *file* exists and is executable.

'-O *file*' True if *file* exists and is owned by the current effective user id.

'-G *file*' True if *file* exists and is owned by the current effective group id.

16.3.3 File characteristic tests

These options test other file characteristics.

'-e *file*' True if *file* exists.

'-s *file*' True if *file* exists and has a size greater than zero.

'*file1* -nt *file2*'
True if *file1* is newer (according to modification date) than *file2*, or if *file1* exists and *file2* does not.

'*file1* -ot *file2*'
True if *file1* is older (according to modification date) than *file2*, or if *file2* exists and *file1* does not.

'*file1* -ef *file2*'
: True if *file1* and *file2* have the same device and inode numbers, i.e., if they are hard links to each other.

16.3.4 String tests

These options test string characteristics. Strings are not quoted for `test`, though you may need to quote them to protect characters with special meaning to the shell, e.g., spaces.

'-z *string*'
: True if the length of *string* is zero.

'-n *string*'
'*string*' True if the length of *string* is nonzero.

'*string1* = *string2*'
: True if the strings are equal.

'*string1* != *string2*'
: True if the strings are not equal.

16.3.5 Numeric tests

Numeric relationals. The arguments must be entirely numeric (possibly negative), or the special expression `-l string`, which evaluates to the length of *string*.

'*arg1* -eq *arg2*'
'*arg1* -ne *arg2*'
'*arg1* -lt *arg2*'
'*arg1* -le *arg2*'
'*arg1* -gt *arg2*'
'*arg1* -ge *arg2*'
: These arithmetic binary operators return true if *arg1* is equal, not-equal, less-than, less-than-or-equal, greater-than, or greater-than-or-equal than *arg2*, respectively.

For example:

```
test -1 -gt -2 && echo yes
⇒ yes
test -l abc -gt 1 && echo yes
⇒ yes
test 0x100 -eq 1
error  test: integer expression expected before -eq
```

16.3.6 Connectives for `test`

The usual logical connectives.

'! *expr*' True if *expr* is false.

'*expr1* `-a` *expr2*'
True if both *expr1* and *expr2* are true.

'*expr1* `-o` *expr2*'
True if either *expr1* or *expr2* is true.

16.4 `expr`: Evaluate expressions

`expr` evaluates an expression and writes the result on standard output. Each token of the expression must be a separate argument.

Operands are either numbers or strings. `expr` converts anything appearing in an operand position to an integer or a string depending on the operation being applied to it.

Strings are not quoted for `expr` itself, though you may need to quote them to protect characters with special meaning to the shell, e.g., spaces.

Operators may be given as infix symbols or prefix keywords. Parentheses may be used for grouping in the usual manner (you must quote parentheses to avoid the shell evaluating them, however).

Exit status:

0 if the expression is neither null nor 0,
1 if the expression is null or 0,
2 for invalid expressions.

16.4.1 String expressions

`expr` supports pattern matching and other string operators. These have lower precedence than both the numeric and relational operators (in the next sections).

'*string* `:` *regex*'
Perform pattern matching. The arguments are converted to strings and the second is considered to be a (basic, a la GNU `grep`) regular expression, with a `^` implicitly prepended. The first argument is then matched against this regular expression.

If the match succeeds and *regex* uses '`\(`' and '`\)`', the `:` expression returns the part of *string* that matched the subexpression; otherwise, it returns the number of characters matched.

If the match fails, the `:` operator returns the null string if '`\(`' and '`\)`' are used in *regex*, otherwise 0.

Only the first '`\( ... \)`' pair is relevant to the return value; additional pairs are meaningful only for grouping the regular expression operators.

In the regular expression, `\+`, `\?`, and `\|` are operators which respectively match one or more, zero or one, or separate alternatives. SunOS and other `expr`'s treat these as regular characters. (POSIX allows either behavior.) See section "Regular Expression Library" in *Regex*, for details of regular expression syntax. Some examples are in Section 16.4.4 [Examples of expr], page 104.

'`match` *string regex*'
: An alternative way to do pattern matching. This is the same as '*string* `:` *regex*'.

'`substr` *string position length*'
: Returns the substring of *string* beginning at *position* with length at most *length*. If either *position* or *length* is negative, zero, or non-numeric, returns the null string.

'`index` *string charset*'
: Returns the first position in *string* where the first character in *charset* was found. If no character in *charset* is found in *string*, return 0.

'`length` *string*'
: Returns the length of *string*.

'`+` *token*'
: Interpret *token* as a string, even if it is a keyword like *match* or an operator like `/`. This makes it possible to test `expr length + "$x"` or `expr + "$x" : '.*/\(.\)'` and have it do the right thing even if the value of *$x* happens to be (for example) `/` or `index`. This operator is a GNU extension. Portable shell scripts should use `" $token" : ' \(.*\)'` instead of `+ "$token"`.

To make `expr` interpret keywords as strings, you must use the `quote` operator.

16.4.2 Numeric expressions

`expr` supports the usual numeric operators, in order of increasing precedence. The string operators (previous section) have lower precedence, the connectives (next section) have higher.

'`+ -`'
: Addition and subtraction. Both arguments are converted to numbers; an error occurs if this cannot be done.

'`* / %`'
: Multiplication, division, remainder. Both arguments are converted to numbers; an error occurs if this cannot be done.

16.4.3 Relations for `expr`

`expr` supports the usual logical connectives and relations. These are higher precedence than either the string or numeric operators (previous sections). Here is the list, lowest-precedence operator first.

'`|`'
: Returns its first argument if that is neither null nor 0, otherwise its second argument.

'`&`'
: Return its first argument if neither argument is null or 0, otherwise 0.

'`< <= = == != >= >`'
: Compare the arguments and return 1 if the relation is true, 0 otherwise. `==` is a synonym for `=`. `expr` first tries to convert both arguments to numbers and do a numeric comparison; if either conversion fails, it does a lexicographic comparison using the character collating sequence specified by the `LC_COLLATE` locale.

16.4.4 Examples of using expr

Here are a few examples, including quoting for shell metacharacters.

To add 1 to the shell variable `foo`, in Bourne-compatible shells:

```
foo=`expr $foo + 1`
```

To print the non-directory part of the file name stored in `$fname`, which need not contain a `/`.

```
expr $fname : '.*/\(.*\)' '|' $fname
```

An example showing that `\+` is an operator:

```
expr aaa : 'a\+'
⇒ 3
expr abc : 'a\(.\)c'
⇒ b
expr index abcdef cz
⇒ 3
expr index index a
error  expr: syntax error
expr index quote index a
⇒ 0
```

17 Redirection

Unix shells commonly provide several forms of *redirection*—ways to change the input source or output destination of a command. But one useful redirection is performed by a separate command, not by the shell; it's described here.

17.1 `tee`: Redirect output to multiple files

The `tee` command copies standard input to standard output and also to any files given as arguments. This is useful when you want not only to send some data down a pipe, but also to save a copy. Synopsis:

```
tee [option]... [file]...
```

If a file being written to does not already exist, it is created. If a file being written to already exists, the data it previously contained is overwritten unless the `-a` option is used.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-a`'
'`--append`'
    Append standard input to the given files rather than overwriting them.

'`-i`'
'`--ignore-interrupts`'
    Ignore interrupt signals.

18 File name manipulation

This section describes commands that manipulate file names.

18.1 `basename`: Strip directory and suffix from a file name

`basename` removes any leading directory components from *name*. Synopsis:

```
basename name [suffix]
```

If *suffix* is specified and is identical to the end of *name*, it is removed from *name* as well. `basename` prints the result on standard output.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

18.2 `dirname`: Strip non-directory suffix from a file name

`dirname` prints all but the final slash-delimited component of a string (presumably a filename). Synopsis:

```
dirname name
```

If *name* is a single component, `dirname` prints '`.`' (meaning the current directory).

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

18.3 `pathchk`: Check file name portability

`pathchk` checks portability of filenames. Synopsis:

```
pathchk [option]... name...
```

For each *name*, `pathchk` prints a message if any of these conditions is true:

1. one of the existing directories in *name* does not have search (execute) permission,
2. the length of *name* is larger than its filesystem's maximum file name length,
3. the length of one component of *name*, corresponding to an existing directory name, is larger than its filesystem's maximum length for a file name component.

The program accepts the following option. Also see Chapter 2 [Common options], page 2.

'`-p`'
'`--portability`'
: Instead of performing length checks on the underlying filesystem, test the length of each file name and its components against the POSIX minimum limits for portability. Also check that the file name contains no characters not in the portable file name character set.

Exit status:

0 if all specified file names passed all of the tests,
1 otherwise.

19 Working context

This section describes commands that display or alter the context in which you are working: the current directory, the terminal settings, and so forth. See also the user-related commands in the next section.

19.1 pwd: Print working directory

`pwd` prints the fully resolved name of the current directory. That is, all components of the printed name will be actual directory names—none will be symbolic links.

Because most shells have a built-in command by the same name, using the unadorned command name in a script or interactively may get you different functionality than that described here.

The only options are a lone '`--help`' or '`--version`'. See Chapter 2 [Common options], page 2.

19.2 stty: Print or change terminal characteristics

`stty` prints or changes terminal characteristics, such as baud rate. Synopses:

```
stty [option] [setting]...
stty [option]
```

If given no line settings, `stty` prints the baud rate, line discipline number (on systems that support it), and line settings that have been changed from the values set by '`stty sane`'. By default, mode reading and setting are performed on the tty line connected to standard input, although this can be modified by the '`--file`' option.

`stty` accepts many non-option arguments that change aspects of the terminal line operation, as described below.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-a`'
'`--all`'
: Print all current settings in human-readable form. This option may not be used in combination with any line settings.

'`-F` *device*'
'`--file=`*device*'
: Set the line opened by the filename specified in *device* instead of the tty line connected to standard input. This option is necessary because opening a POSIX tty requires use of the `O_NONDELAY` flag to prevent a POSIX tty from blocking until the carrier detect line is high if the `clocal` flag is not set. Hence, it is not always possible to allow the shell to open the device in the traditional manner.

'`-g`'
'`--save`'
: Print all current settings in a form that can be used as an argument to another `stty` command to restore the current settings. This option may not be used in combination with any line settings.

Many settings can be turned off by preceding them with a '-'. Such arguments are marked below with "May be negated" in their description. The descriptions themselves refer to the positive case, that is, when *not* negated (unless stated otherwise, of course).

Some settings are not available on all POSIX systems, since they use extensions. Such arguments are marked below with "Non-POSIX" in their description. On non-POSIX systems, those or other settings also may not be available, but it's not feasible to document all the variations: just try it and see.

19.2.1 Control settings

Control settings:

'`parenb`' Generate parity bit in output and expect parity bit in input. May be negated.

'`parodd`' Set odd parity (even if negated). May be negated.

'`cs5`'
'`cs6`'
'`cs7`'
'`cs8`' Set character size to 5, 6, 7, or 8 bits.

'`hup`'
'`hupcl`' Send a hangup signal when the last process closes the tty. May be negated.

'`cstopb`' Use two stop bits per character (one if negated). May be negated.

'`cread`' Allow input to be received. May be negated.

'`clocal`' Disable modem control signals. May be negated.

'`crtscts`' Enable RTS/CTS flow control. Non-POSIX. May be negated.

19.2.2 Input settings

'`ignbrk`' Ignore break characters. May be negated.

'`brkint`' Make breaks cause an interrupt signal. May be negated.

'`ignpar`' Ignore characters with parity errors. May be negated.

'`parmrk`' Mark parity errors (with a 255-0-character sequence). May be negated.

'`inpck`' Enable input parity checking. May be negated.

'`istrip`' Clear high (8th) bit of input characters. May be negated.

'`inlcr`' Translate newline to carriage return. May be negated.

'`igncr`' Ignore carriage return. May be negated.

'`icrnl`' Translate carriage return to newline. May be negated.

'`ixon`' Enable XON/XOFF flow control (that is, *CTRL-S*/*CTRL-Q*). May be negated.

'`ixoff`'
'`tandem`' Enable sending of `stop` character when the system input buffer is almost full, and `start` character when it becomes almost empty again. May be negated.

'iuclc' Translate uppercase characters to lowercase. Non-POSIX. May be negated.

'ixany' Allow any character to restart output (only the start character if negated). Non-POSIX. May be negated.

'imaxbel' Enable beeping and not flushing input buffer if a character arrives when the input buffer is full. Non-POSIX. May be negated.

19.2.3 Output settings

These arguments specify output-related operations.

'opost' Postprocess output. May be negated.

'olcuc' Translate lowercase characters to uppercase. Non-POSIX. May be negated.

'ocrnl' Translate carriage return to newline. Non-POSIX. May be negated.

'onlcr' Translate newline to carriage return-newline. Non-POSIX. May be negated.

'onocr' Do not print carriage returns in the first column. Non-POSIX. May be negated.

'onlret' Newline performs a carriage return. Non-POSIX. May be negated.

'ofill' Use fill (padding) characters instead of timing for delays. Non-POSIX. May be negated.

'ofdel' Use delete characters for fill instead of null characters. Non-POSIX. May be negated.

'nl1'
'nl0' Newline delay style. Non-POSIX.

'cr3'
'cr2'
'cr1'
'cr0' Carriage return delay style. Non-POSIX.

'tab3'
'tab2'
'tab1'
'tab0' Horizontal tab delay style. Non-POSIX.

'bs1'
'bs0' Backspace delay style. Non-POSIX.

'vt1'
'vt0' Vertical tab delay style. Non-POSIX.

'ff1'
'ff0' Form feed delay style. Non-POSIX.

19.2.4 Local settings

'isig' Enable `interrupt`, `quit`, and `suspend` special characters. May be negated.

'icanon' Enable `erase`, `kill`, `werase`, and `rprnt` special characters. May be negated.

'iexten' Enable non-POSIX special characters. May be negated.

'echo' Echo input characters. May be negated.

'echoe'
'crterase'
Echo `erase` characters as backspace-space-backspace. May be negated.

'echok' Echo a newline after a `kill` character. May be negated.

'echonl' Echo newline even if not echoing other characters. May be negated.

'noflsh' Disable flushing after `interrupt` and `quit` special characters. May be negated.

'xcase' Enable input and output of uppercase characters by preceding their lowercase equivalents with '\', when `icanon` is set. Non-POSIX. May be negated.

'tostop' Stop background jobs that try to write to the terminal. Non-POSIX. May be negated.

'echoprt'
'prterase'
Echo erased characters backward, between '\' and '/'. Non-POSIX. May be negated.

'echoctl'
'ctlecho' Echo control characters in hat notation ('^c') instead of literally. Non-POSIX. May be negated.

'echoke'
'crtkill' Echo the `kill` special character by erasing each character on the line as indicated by the `echoprt` and `echoe` settings, instead of by the `echoctl` and `echok` settings. Non-POSIX. May be negated.

19.2.5 Combination settings

Combination settings:

'evenp'
'parity' Same as `parenb -parodd cs7`. May be negated. If negated, same as `-parenb cs8`.

'oddp' Same as `parenb parodd cs7`. May be negated. If negated, same as `-parenb cs8`.

'nl' Same as `-icrnl -onlcr`. May be negated. If negated, same as `icrnl -inlcr -igncr onlcr -ocrnl -onlret`.

'ek' Reset the `erase` and `kill` special characters to their default values.

'sane' Same as:

```
cread -ignbrk brkint -inlcr -igncr icrnl -ixoff
-iuclc -ixany imaxbel opost -olcuc -ocrnl onlcr
-onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0
ff0 isig icanon iexten echo echoe echok -echonl
-noflsh -xcase -tostop -echoprt echoctl echoke
```

and also sets all special characters to their default values.

'cooked' Same as `brkint ignpar istrip icrnl ixon opost isig icanon`, plus sets the `eof` and `eol` characters to their default values if they are the same as the `min` and `time` characters. May be negated. If negated, same as `raw`.

'raw' Same as:

```
-ignbrk -brkint -ignpar -parmrk -inpck -istrip
-inlcr -igncr -icrnl -ixon -ixoff -iuclc -ixany
-imaxbel -opost -isig -icanon -xcase min 1 time 0
```

May be negated. If negated, same as `cooked`.

'cbreak' Same as `-icanon`. May be negated. If negated, same as `icanon`.

'pass8' Same as `-parenb -istrip cs8`. May be negated. If negated, same as `parenb istrip cs7`.

'litout' Same as `-parenb -istrip -opost cs8`. May be negated. If negated, same as `parenb istrip opost cs7`.

'decctlq' Same as `-ixany`. Non-POSIX. May be negated.

'tabs' Same as `tab0`. Non-POSIX. May be negated. If negated, same as `tab3`.

'lcase'
'LCASE' Same as `xcase iuclc olcuc`. Non-POSIX. May be negated.

'crt' Same as `echoe echoctl echoke`.

'dec' Same as `echoe echoctl echoke -ixany intr ^C erase ^? kill C-u`.

19.2.6 Special characters

The special characters' default values vary from system to system. They are set with the syntax '`name value`', where the names are listed below and the value can be given either literally, in hat notation ('`^c`'), or as an integer which may start with '`0x`' to indicate hexadecimal, '`0`' to indicate octal, or any other digit to indicate decimal.

For GNU stty, giving a value of `^-` or `undef` disables that special character. (This is incompatible with Ultrix `stty`, which uses a value of '`u`' to disable a special character. GNU `stty` treats a value '`u`' like any other, namely to set that special character to U.)

'intr' Send an interrupt signal.

'quit' Send a quit signal.

'erase' Erase the last character typed.

'kill' Erase the current line.

'eof' Send an end of file (terminate the input).

'eol' End the line.

'eol2' Alternate character to end the line. Non-POSIX.

'swtch' Switch to a different shell layer. Non-POSIX.

'start' Restart the output after stopping it.

'stop' Stop the output.

'susp' Send a terminal stop signal.

'dsusp' Send a terminal stop signal after flushing the input. Non-POSIX.

'rprnt' Redraw the current line. Non-POSIX.

'werase' Erase the last word typed. Non-POSIX.

'lnext' Enter the next character typed literally, even if it is a special character. Non-POSIX.

19.2.7 Special settings

'min *n*' Set the minimum number of characters that will satisfy a read until the time value has expired, when `-icanon` is set.

'time *n*' Set the number of tenths of a second before reads time out if the minimum number of characters have not been read, when `-icanon` is set.

'ispeed *n*'
Set the input speed to *n*.

'ospeed *n*'
Set the output speed to *n*.

'rows *n*' Tell the tty kernel driver that the terminal has *n* rows. Non-POSIX.

'cols *n*'
'columns *n*'
Tell the kernel that the terminal has *n* columns. Non-POSIX.

'size' Print the number of rows and columns that the kernel thinks the terminal has. (Systems that don't support rows and columns in the kernel typically use the environment variables `LINES` and `COLUMNS` instead; however, GNU `stty` does not know anything about them.) Non-POSIX.

'line *n*' Use line discipline *n*. Non-POSIX.

'speed' Print the terminal speed.

'*n*' Set the input and output speeds to *n*. *n* can be one of: 0 50 75 110 134 134.5 150 200 300 600 1200 1800 2400 4800 9600 19200 38400 `exta` `extb`. `exta` is the same as 19200; `extb` is the same as 38400. 0 hangs up the line if `-clocal` is set.

19.3 `printenv`: Print all or some environment variables

`printenv` prints environment variable values. Synopsis:

```
printenv [option] [variable]...
```

If no *variables* are specified, `printenv` prints the value of every environment variable. Otherwise, it prints the value of each *variable* that is set, and nothing for those that are not set.

The only options are a lone '`--help`' or '`--version`'. See Chapter 2 [Common options], page 2.

Exit status:

0 if all variables specified were found
1 if at least one specified variable was not found
2 if a write error occurred

19.4 `tty`: Print file name of terminal on standard input

`tty` prints the file name of the terminal connected to its standard input. It prints '`not a tty`' if standard input is not a terminal. Synopsis:

```
tty [option]...
```

The program accepts the following option. Also see Chapter 2 [Common options], page 2.

'`-s`'
'`--silent`'
'`--quiet`' Print nothing; only return an exit status.

Exit status:

0 if standard input is a terminal
1 if standard input is not a terminal
2 if given incorrect arguments
3 if a write error occurs

20 User information

This section describes commands that print user-related information: logins, groups, and so forth.

20.1 `id`: Print real and effective uid and gid

`id` prints information about the given user, or the process running it if no user is specified. Synopsis:

```
id [option]... [username]
```

By default, it prints the real user id, real group id, effective user id if different from the real user id, effective group id if different from the real group id, and supplemental group ids.

Each of these numeric values is preceded by an identifying string and followed by the corresponding user or group name in parentheses.

The options cause `id` to print only part of the above information. Also see Chapter 2 [Common options], page 2.

'`-g`'
'`--group`' Print only the group id.

'`-G`'
'`--groups`'
Print only the supplementary groups.

'`-n`'
'`--name`' Print the user or group name instead of the ID number. Requires `-u`, `-g`, or `-G`.

'`-r`'
'`--real`' Print the real, instead of effective, user or group id. Requires `-u`, `-g`, or `-G`.

'`-u`'
'`--user`' Print only the user id.

20.2 `logname`: Print current login name

`logname` prints the calling user's name, as found in the file '`/etc/utmp`', and exits with a status of 0. If there is no '`/etc/utmp`' entry for the calling process, `logname` prints an error message and exits with a status of 1.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

20.3 `whoami`: Print effective user id

`whoami` prints the user name associated with the current effective user id. It is equivalent to the command '`id -un`'.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

20.4 `groups`: Print group names a user is in

`groups` prints the names of the primary and any supplementary groups for each given *username*, or the current process if no names are given. If names are given, the name of each user is printed before the list of that user's groups. Synopsis:

```
groups [username]...
```

The group lists are equivalent to the output of the command '`id -Gn`'.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

20.5 `users`: Print login names of users currently logged in

`users` prints on a single line a blank-separated list of user names of users currently logged in to the current host. Each user name corresponds to a login session, so if a user has more than one login session, that user's name will appear the same number of times in the output. Synopsis:

```
users [file]
```

With no *file* argument, `users` extracts its information from the file '`/etc/utmp`'. If a file argument is given, `users` uses that file instead. A common choice is '`/etc/wtmp`'.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

20.6 `who`: Print who is currently logged in

`who` prints information about users who are currently logged on. Synopsis:

```
who [option] [file] [am i]
```

If given no non-option arguments, `who` prints the following information for each user currently logged on: login name, terminal line, login time, and remote hostname or X display.

If given one non-option argument, `who` uses that instead of '`/etc/utmp`' as the name of the file containing the record of users logged on. '`/etc/wtmp`' is commonly given as an argument to `who` to look at who has previously logged on.

If given two non-option arguments, `who` prints only the entry for the user running it (determined from its standard input), preceded by the hostname. Traditionally, the two arguments given are '`am i`', as in '`who am i`'.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-m`'
: Same as '`who am i`'.

'`-q`'
'`--count`'
: Print only the login names and the number of users logged on. Overrides all other options.

'`-s`'
: Ignored; for compatibility with other versions of `who`.

'-i'
'-u'
'--idle' After the login time, print the number of hours and minutes that the user has been idle. '.' means the user was active in last minute. 'old' means the user was idle for more than 24 hours.

'-l'
'--lookup'
Attempt to canonicalize hostnames found in utmp through a DNS lookup. This is not the default because it can cause significant delays on systems with automatic dial-up internet access.

'-H'
'--heading'
Print a line of column headings.

'-w'
'-T'
'--mesg'
'--message'
'--writable'
After each login name print a character indicating the user's message status:

'+' allowing `write` messages
'-' disallowing `write` messages
'?' cannot find terminal device

21 System context

This section describes commands that print or change system-wide information.

21.1 `date`: Print or set system date and time

Synopses:

```
date [option]... [+format]
date [-u|--utc|--universal] [ MMDDhhmm[[CC]YY][.ss] ]
```

Invoking `date` with no *format* argument is equivalent to invoking '`date '+%a %b %e %H:%M:%S %Z %Y'`'.

If given an argument that starts with a '`+`', `date` prints the current time and date (or the time and date specified by the `--date` option, see below) in the format defined by that argument, which is the same as in the `strftime` function. Except for directives, which start with '`%`', characters in the format string are printed unchanged. The directives are described below.

21.1.1 Time directives

`date` directives related to times.

'`%H`' hour (00...23)

'`%I`' hour (01...12)

'`%k`' hour (0...23)

'`%l`' hour (1...12)

'`%M`' minute (00...59)

'`%N`' nanoseconds (000000000...999999999)

'`%p`' locale's upper case '`AM`' or '`PM`' (blank in many locales)

'`%P`' locale's lower case '`am`' or '`pm`' (blank in many locales)

'`%r`' time, 12-hour (hh:mm:ss [AP]M)

'`%R`' time, 24-hour (hh:mm). Same as `%H:%M`.

'`%s`' seconds since the epoch, i.e., 1 January 1970 00:00:00 UTC (a GNU extension). Note that this value is the number of seconds between the epoch and the current date as defined by the localtime system call. It isn't changed by the '`--date`' option.

'`%S`' second (00...60). The range is [00...60], and not [00...59], in order to accommodate the occasional positive leap second.

'`%T`' time, 24-hour (hh:mm:ss)

'`%X`' locale's time representation (%H:%M:%S)

'%z' RFC-822 style numeric time zone (e.g., -0600 or +0100), or nothing if no time zone is determinable. This value reflects the *current* time zone. It isn't changed by the '--date' option.

'%Z' time zone (e.g., EDT), or nothing if no time zone is determinable. Note that this value reflects the *current* time zone. It isn't changed by the '--date' option.

21.1.2 Date directives

`date` directives related to dates.

'%a' locale's abbreviated weekday name (Sun...Sat)

'%A' locale's full weekday name, variable length (Sunday...Saturday)

'%b' locale's abbreviated month name (Jan...Dec)

'%B' locale's full month name, variable length (January...December)

'%c' locale's date and time (Sat Nov 04 12:02:33 EST 1989)

'%C' century (year divided by 100 and truncated to an integer) (00...99)

'%d' day of month (01...31)

'%D' date (mm/dd/yy)

'%e' blank-padded day of month (1...31)

'%F' the ISO 8601 standard date format: `%Y-%m-%d`. This is the preferred form for all uses.

'%g' The year corresponding to the ISO week number, but without the century (range `00` through `99`). This has the same format and value as `%y`, except that if the ISO week number (see `%V`) belongs to the previous or next year, that year is used instead.

'%G' The year corresponding to the ISO week number. This has the same format and value as `%Y`, except that if the ISO week number (see `%V`) belongs to the previous or next year, that year is used instead.

'%h' same as %b

'%j' day of year (001...366)

'%m' month (01...12)

'%u' day of week (1...7) with 1 corresponding to Monday

'%U' week number of year with Sunday as first day of week (00...53). Days in a new year preceding the first Sunday are in week zero.

'%V' week number of year with Monday as first day of the week as a decimal (01...53). If the week containing January 1 has four or more days in the new year, then it is considered week 1; otherwise, it is week 53 of the previous year, and the next week is week 1. (See the ISO 8601 standard.)

'%w' day of week (0...6) with 0 corresponding to Sunday

'%W' week number of year with Monday as first day of week (00...53). Days in a new year preceding the first Monday are in week zero.

'%x' locale's date representation (mm/dd/yy)

'%y' last two digits of year (00...99)

'%Y' year (1970....)

21.1.3 Literal directives

`date` directives that produce literal strings.

'%%' a literal %

'%n' a newline

'%t' a horizontal tab

21.1.4 Padding

By default, `date` pads numeric fields with zeroes, so that, for example, numeric months are always output as two digits. GNU `date` recognizes the following numeric modifiers between the '%' and the directive.

'-' (hyphen) do not pad the field; useful if the output is intended for human consumption.

'_' (underscore) pad the field with spaces; useful if you need a fixed number of characters in the output, but zeroes are too distracting.

These are GNU extensions.

Here is an example illustrating the differences:

```
date +%d/%m -d "Feb 1"
⇒ 01/02
date +%-d/%-m -d "Feb 1"
⇒ 1/2
date +%_d/%_m -d "Feb 1"
⇒  1/ 2
```

21.1.5 Setting the time

If given an argument that does not start with '+', `date` sets the system clock to the time and date specified by that argument (as described below). You must have appropriate privileges to set the system clock. The '`--date`' and '`--set`' options may not be used with such an argument. The '`--universal`' option may be used with such an argument to indicate that the specified time and date are relative to Coordinated Universal Time rather than to the local time zone.

The argument must consist entirely of digits, which have the following meaning:

'MM' month

'DD' day within month

'hh' hour

'mm' minute

'CC' first two digits of year (optional)

'YY' last two digits of year (optional)

'ss' second (optional)

The '--set' option also sets the system clock; see the next section.

21.1.6 Options for date

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-d *datestr*'
'--date=*datestr*'
: Display the time and date specified in *datestr* instead of the current time and date. *datestr* can be in almost any common format. It can contain month names, time zones, 'am' and 'pm', 'yesterday', 'ago', 'next', etc. See Chapter 27 [Date input formats], page 141.

'-f *datefile*'
'--file=*datefile*'
: Parse each line in *datefile* as with '-d' and display the resulting time and date. If *datefile* is '-', use standard input. This is useful when you have many dates to process, because the system overhead of starting up the date executable many times can be considerable.

'-I *timespec*'
'--iso-8601[=*timespec*]'
: Display the date using the ISO 8601 format, '%Y-%m-%d'.

The argument *timespec* specifies the number of additional terms of the time to include. It can be one of the following:

'auto' The default behavior: print just the date.

'hours' Append the hour of the day to the date.

'minutes' Append the hours and minutes.

'seconds' Append the hours, minutes, and seconds.

If showing any time terms, then include the time zone using the format '%z'.

If *timespec* is omitted with '--iso-8601', the default is 'auto'. On older systems, GNU date instead supports an obsolete option '-I[*timespec*]', where *timespec* defaults to 'auto'. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow '-I' without an argument; use '--iso-8601' instead.

'-R'
'--rfc-822'
: Display the time and date using the RFC-822-conforming format, '%a, %_d %b %Y %H:%M:%S %z'.

'-r *file*'
'--reference=*file*'
: Display the time and date reference according to the last modification time of *file*, instead of the current time and date.

'-s *datestr*'
'--set=*datestr*'
: Set the time and date to *datestr*. See '-d' above.

'-u'
'--utc'
'--universal'
: Use Coordinated Universal Time (UTC) by operating as if the TZ environment variable were set to the string 'UTC0'. Normally, date operates in the time zone indicated by TZ, or the system default if TZ is not set. Coordinated Universal Time is often called "Greenwich Mean Time" (GMT) for historical reasons.

21.1.7 Examples of date

Here are a few examples. Also see the documentation for the '-d' option in the previous section.

- To print the date of the day before yesterday:
```
date --date='2 days ago'
```
- To print the date of the day three months and one day hence:
```
date --date='3 months 1 day'
```
- To print the day of year of Christmas in the current year:
```
date --date='25 Dec' +%j
```
- To print the current full month name and the day of the month:
```
date '+%B %d'
```
  But this may not be what you want because for the first nine days of the month, the '%d' expands to a zero-padded two-digit field, for example 'date -d 1may '+%B %d'' will print 'May 01'.
- To print a date without the leading zero for one-digit days of the month, you can use the (GNU extension) - modifier to suppress the padding altogether.
```
date -d 1may '+%B %-d
```
- To print the current date and time in the format required by many non-GNU versions of date when setting the system clock:
```
date +%m%d%H%M%Y.%S
```
- To set the system clock forward by two minutes:
```
date --set='+2 minutes'
```

- To print the date in the format specified by RFC-822, use '`date --rfc`'. I just did and saw this:

  ```
  Mon, 25 Mar 1996 23:34:17 -0600
  ```

- To convert a date string to the number of seconds since the epoch (which is 1970-01-01 00:00:00 UTC), use the '`--date`' option with the '`%s`' format. That can be useful in sorting and/or graphing and/or comparing data by date. The following command outputs the number of the seconds since the epoch for the time two minutes after the epoch:

  ```
  date --date='1970-01-01 00:02:00 +0000' +%s
  120
  ```

  If you do not specify time zone information in the date string, `date` uses your computer's idea of the time zone when interpreting the string. For example, if your computer's time zone is that of Cambridge, Massachusetts, which was then 5 hours (i.e., 18,000 seconds) behind UTC:

  ```
  # local time zone used
  date --date='1970-01-01 00:02:00' +%s
  18120
  ```

- If you're sorting or graphing dated data, your raw date values may be represented as seconds since the epoch. But few people can look at the date '`946684800`' and casually note "Oh, that's the first second of the year 2000 in Greenwich, England."

  ```
  date --date='2000-01-01 UTC' +%s
  946684800
  ```

  To convert such an unwieldy number of seconds back to a more readable form, use a command like this:

  ```
  # local time zone used
  date -d '1970-01-01 UTC 946684800 seconds' +"%Y-%m-%d %T %z"
  1999-12-31 19:00:00 -0500
  ```

21.2 `uname`: Print system information

`uname` prints information about the machine and operating system it is run on. If no options are given, `uname` acts as if the `-s` option were given. Synopsis:

```
uname [option]...
```

If multiple options or `-a` are given, the selected information is printed in this order:

`kernel-name nodename kernel-release kernel-version machine processor hardware-platform operating-system`

The information may contain internal spaces, so such output cannot be parsed reliably. In the following example, *release* is '`2.2.18ss.e820-bda652a #4 SMP Tue Jun 5 11:24:08 PDT 2001`':

```
uname -a
⇒ Linux dum 2.2.18ss.e820-bda652a #4 SMP Tue Jun 5 11:24:08 PDT 2001 i686 unknown unknown GNU/Linux
```

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-a'
'--all' Print all of the below information.

'-i'
'--hardware-platform'
Print the hardware platform name (sometimes called the hardware implementation).

'-m'
'--machine'
Print the machine hardware name (sometimes called the hardware class).

'-n'
'--nodename'
Print the network node hostname.

'-p'
'--processor'
Print the processor type (sometimes called the instruction set architecture or ISA).

'-o'
'--operating-system'
Print the name of the operating system.

'-r'
'--kernel-release'
Print the kernel release.

'-s'
'--kernel-name'
Print the kernel name.

'-v'
'--kernel-version'
Print the kernel version.

21.3 `hostname`: Print or set system name

With no arguments, `hostname` prints the name of the current host system. With one argument, it sets the current host name to the specified string. You must have appropriate privileges to set the host name. Synopsis:

```
hostname [name]
```

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

21.4 `hostid`: Print numeric host identifier.

`hostid` prints the numeric identifier of the current host in hexadecimal. This command accepts no arguments. The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

For example, here's what it prints on one system I use:

```
$ hostid
1bac013d
```

On that system, the 32-bit quantity happens to be closely related to the system's Internet address, but that isn't always the case.

22 Modified command invocation

This section describes commands that run other commands in some context different than the current one: a modified environment, as a different user, etc.

22.1 `chroot`: Run a command with a different root directory

`chroot` runs a command with a specified root directory. On many systems, only the super-user can do this. Synopses:

```
chroot newroot [command [args]...]
chroot option
```

Ordinarily, filenames are looked up starting at the root of the directory structure, i.e., '`/`'. `chroot` changes the root to the directory *newroot* (which must exist) and then runs *command* with optional *args*. If *command* is not specified, the default is the value of the `SHELL` environment variable or `/bin/sh` if not set, invoked with the '`-i`' option.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

Here are a few tips to help avoid common problems in using chroot. To start with a simple example, make *command* refer to a statically linked binary. If you were to use a dynamically linked executable, then you'd have to arrange to have the shared libraries in the right place under your new root directory.

For example, if you create a statically linked 'ls' executable, and put it in /tmp/empty, you can run this command as root:

```
$ chroot /tmp/empty /ls -Rl /
```

Then you'll see output like this:

```
/:
total 1023
-rwxr-xr-x    1 0        0          1041745 Aug 16 11:17 ls
```

If you want to use a dynamically linked executable, say `bash`, then first run '`ldd bash`' to see what shared objects it needs. Then, in addition to copying the actual binary, also copy the listed files to the required positions under your intended new root directory. Finally, if the executable requires any other files (e.g., data, state, device files), copy them into place, too.

22.2 `env`: Run a command in a modified environment

`env` runs a command with a modified environment. Synopses:

```
env [option]... [name=value]... [command [args]...]
env
```

Arguments of the form '*`variable=value`*' set the environment variable *variable* to value *value*. *value* may be empty ('*`variable=`*'). Setting a variable to an empty value is different from unsetting it.

The first remaining argument specifies the program name to invoke; it is searched for according to the `PATH` environment variable. Any remaining arguments are passed as arguments to that program.

If no command name is specified following the environment specifications, the resulting environment is printed. This is like specifying a command name of `printenv`.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-u` *name*'
'`--unset=`*name*'
: Remove variable *name* from the environment, if it was in the environment.

'`-`'
'`-i`'
'`--ignore-environment`'
: Start with an empty environment, ignoring the inherited environment.

22.3 `nice`: Run a command with modified scheduling priority

`nice` prints or modifies the scheduling priority of a job. Synopsis:

```
nice [option]... [command [arg]...]
```

If no arguments are given, `nice` prints the current scheduling priority, which it inherited. Otherwise, `nice` runs the given *command* with its scheduling priority adjusted. If no *adjustment* is given, the priority of the command is incremented by 10. You must have appropriate privileges to specify a negative adjustment. The priority can be adjusted by `nice` over the range of -20 (the highest priority) to 19 (the lowest).

Because most shells have a built-in command by the same name, using the unadorned command name in a script or interactively may get you different functionality than that described here.

The program accepts the following option. Also see Chapter 2 [Common options], page 2.

'`-n` *adjustment*'
'`--adjustment=`*adjustment*'
: Add *adjustment* instead of 10 to the command's priority.

  On older systems, `nice` supports an obsolete option '`-`*adjustment*'. POSIX 1003.1-2001 (see Section 2.5 [Standards conformance], page 6) does not allow this; use '`-n` *adjustment*' instead.

22.4 `nohup`: Run a command immune to hangups

`nohup` runs the given *command* with hangup signals ignored, so that the command can continue running in the background after you log out. Synopsis:

```
nohup command [arg]...
```

If standard output is a terminal, it is redirected so that it is appended to the file 'nohup.out'; if that cannot be written to, it is appended to the file '$HOME/nohup.out'. If that cannot be written to, the command is not run.

If nohup creates either 'nohup.out' or '$HOME/nohup.out', it creates it with no "group" or "other" access permissions. It does not change the permissions if the output file already existed.

If standard error is a terminal, it is redirected to the same file descriptor as the standard output.

nohup does not automatically put the command it runs in the background; you must do that explicitly, by ending the command line with an '&'. Also, nohup does not change the scheduling priority of *command*; use nice for that, e.g., 'nohup nice *command*'.

The only options are '--help' and '--version'. See Chapter 2 [Common options], page 2.

Exit status:

126 if *command* was found but could not be invoked
127 if nohup itself failed or if *command* could not be found
the exit status of *command* otherwise

22.5 su: Run a command with substitute user and group id

su allows one user to temporarily become another user. It runs a command (often an interactive shell) with the real and effective user id, group id, and supplemental groups of a given *user*. Synopsis:

```
su [option]... [user [arg]...]
```

If no *user* is given, the default is root, the super-user. The shell to use is taken from *user*'s passwd entry, or '/bin/sh' if none is specified there. If *user* has a password, su prompts for the password unless run by a user with effective user id of zero (the super-user).

By default, su does not change the current directory. It sets the environment variables HOME and SHELL from the password entry for *user*, and if *user* is not the super-user, sets USER and LOGNAME to *user*. By default, the shell is not a login shell.

Any additional *arg*s are passed as additional arguments to the shell.

GNU su does not treat '/bin/sh' or any other shells specially (e.g., by setting argv[0] to '-su', passing -c only to certain shells, etc.).

su can optionally be compiled to use syslog to report failed, and optionally successful, su attempts. (If the system supports syslog.) However, GNU su does not check if the user is a member of the wheel group; see below.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'-c *command*'
'--command=*command*'
: Pass *command*, a single command line to run, to the shell with a -c option instead of starting an interactive shell.

'-f'
'--fast' Pass the -f option to the shell. This probably only makes sense if the shell run is csh or tcsh, for which the -f option prevents reading the startup file ('.cshrc'). With Bourne-like shells, the -f option disables file name pattern expansion (globbing), which is not likely to be useful.

'-'
'-l'
'--login' Make the shell a login shell. This means the following. Unset all environment variables except TERM, HOME, and SHELL (which are set as described above), and USER and LOGNAME (which are set, even for the super-user, as described above), and set PATH to a compiled-in default value. Change to *user*'s home directory. Prepend '-' to the shell's name, intended to make it read its login startup file(s).

'-m'
'-p'
'--preserve-environment'
Do not change the environment variables HOME, USER, LOGNAME, or SHELL. Run the shell given in the environment variable SHELL instead of the shell from *user*'s passwd entry, unless the user running su is not the superuser and *user*'s shell is restricted. A *restricted shell* is one that is not listed in the file '/etc/shells', or in a compiled-in list if that file does not exist. Parts of what this option does can be overridden by --login and --shell.

'-s *shell*'
'--shell=*shell*'
Run *shell* instead of the shell from *user*'s passwd entry, unless the user running su is not the superuser and *user*'s shell is restricted (see '-m' just above).

Why GNU su does not support the 'wheel' group

(This section is by Richard Stallman.)

Sometimes a few of the users try to hold total power over all the rest. For example, in 1984, a few users at the MIT AI lab decided to seize power by changing the operator password on the Twenex system and keeping it secret from everyone else. (I was able to thwart this coup and give power back to the users by patching the kernel, but I wouldn't know how to do that in Unix.)

However, occasionally the rulers do tell someone. Under the usual su mechanism, once someone learns the root password who sympathizes with the ordinary users, he or she can tell the rest. The "wheel group" feature would make this impossible, and thus cement the power of the rulers.

I'm on the side of the masses, not that of the rulers. If you are used to supporting the bosses and sysadmins in whatever they do, you might find this idea strange at first.

23 Process control

23.1 `kill`: Send a signal to processes

The `kill` command sends a signal to processes, causing them to terminate or otherwise act upon receiving the signal in some way. Alternatively, it lists information about signals. Synopses:

```
kill [-s signal | --signal signal | -signal] pid...
kill [-l | --list | -t | --table] [signal]...
```

The first form of the `kill` command sends a signal to all *pid* arguments. The default signal to send if none is specified is '`TERM`'. The special signal number '`0`' does not denote a valid signal, but can be used to test whether the *pid* arguments specify processes to which a signal could be sent.

If *pid* is positive, the signal is sent to the process with the process id *pid*. If *pid* is zero, the signal is sent to all processes in the process group of the current process. If *pid* is -1, the signal is sent to all processes for which the user has permission to send a signal. If *pid* is less than -1, the signal is sent to all processes in the process group that equals the absolute value of *pid*.

If *pid* is not positive, a system-dependent set of system processes is excluded from the list of processes to which the signal is sent.

If a negative *PID* argument is desired as the first one, either a signal must be specified as well, or the option parsing must be interrupted with '–' before the first *pid* argument. The following three commands are equivalent:

```
kill -15 -1
kill -TERM -1
kill -- -1
```

The first form of the `kill` command succeeds if every *pid* argument specifies at least one process that the signal was sent to.

The second form of the `kill` command lists signal information. Either the '`-l`' or '`--list`' option, or the '`-t`' or '`--table`' option must be specified. Without any *signal* argument, all supported signals are listed. The output of '`-l`' or '`--list`' is a list of the signal names, one per line; if *signal* is already a name, the signal number is printed instead. The output of '`-t`' or '`--table`' is a table of signal numbers, names, and descriptions. This form of the `kill` command succeeds if all *signal* arguments are valid and if there is no output error.

The `kill` command also supports the '`--help`' and '`--version`' options. See Chapter 2 [Common options], page 2.

A *signal* may be a signal name like '`HUP`', or a signal number like '`1`', or an exit status of a process terminated by the signal. A signal name can be given in canonical form or prefixed by '`SIG`'. The case of the letters is ignored, except for the '`-signal`' option which must use upper case to avoid ambiguity with lower case option letters. The following signal names and numbers are supported on all POSIX compliant systems:

'`HUP`' 1. Hangup.

'INT' 2. Terminal interrupt.

'QUIT' 3. Terminal quit.

'ABRT' 6. Process abort.

'KILL' 9. Kill (cannot be caught or ignored).

'ALRM' 14. Alarm Clock.

'TERM' 15. Termination.

Other supported signal names have system-dependent corresponding numbers. All systems conforming to POSIX 1003.1-2001 also support the following signals:

'BUS' Access to an undefined portion of a memory object.

'CHLD' Child process terminated, stopped, or continued.

'CONT' Continue executing, if stopped.

'FPE' Erroneous arithmetic operation.

'ILL' Illegal Instruction.

'PIPE' Write on a pipe with no one to read it.

'SEGV' Invalid memory reference.

'STOP' Stop executing (cannot be caught or ignored).

'TSTP' Terminal stop.

'TTIN' Background process attempting read.

'TTOU' Background process attempting write.

'URG' High bandwidth data is available at a socket.

'USR1' User-defined signal 1.

'USR2' User-defined signal 2.

POSIX 1003.1-2001 systems that support the XSI extension also support the following signals:

'POLL' Pollable event.

'PROF' Profiling timer expired.

'SYS' Bad system call.

'TRAP' Trace/breakpoint trap.

'VTALRM' Virtual timer expired.

'XCPU' CPU time limit exceeded.

'XFSZ' File size limit exceeded.

POSIX 1003.1-2001 systems that support the XRT extension also support at least eight real-time signals called 'RTMIN', 'RTMIN+1', . . . , 'RTMAX-1', 'RTMAX'.

24 Delaying

24.1 `sleep`: Delay for a specified time

`sleep` pauses for an amount of time specified by the sum of the values of the command line arguments. Synopsis:

```
sleep number[smhd]...
```

Each argument is a number followed by an optional unit; the default is seconds. The units are:

'`s`' seconds

'`m`' minutes

'`h`' hours

'`d`' days

Historical implementations of `sleep` have required that *number* be an integer. However, GNU `sleep` accepts arbitrary floating point numbers.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

25 Numeric operations

These programs do numerically-related operations.

25.1 `factor`: Print prime factors

`factor` prints prime factors. Synopses:

```
factor [number]...
factor option
```

If no *number* is specified on the command line, `factor` reads numbers from standard input, delimited by newlines, tabs, or spaces.

The only options are '`--help`' and '`--version`'. See Chapter 2 [Common options], page 2.

The algorithm it uses is not very sophisticated, so for some inputs `factor` runs for a long time. The hardest numbers to factor are the products of large primes. Factoring the product of the two largest 32-bit prime numbers takes over 10 minutes of CPU time on a 400MHz Pentium II.

```
$ p=`echo '4294967279 * 4294967291'|bc`
$ factor $p
18446743979220271189: 4294967279 4294967291
```

In contrast, `factor` factors the largest 64-bit number in just over a tenth of a second:

```
$ factor `echo '2^64-1'|bc`
18446744073709551615: 3 5 17 257 641 65537 6700417
```

25.2 `seq`: Print numeric sequences

`seq` prints a sequence of numbers to standard output. Synopses:

```
seq [option]... [first [increment]] last...
```

`seq` prints the numbers from *first* to *last* by *increment*. By default, *first* and *increment* are both 1, and each number is printed on its own line. All numbers can be reals, not just integers.

The program accepts the following options. Also see Chapter 2 [Common options], page 2.

'`-f` *format*'
'`--format=`*format*'
: Print all numbers using *format*; default '`%g`'. *format* must contain exactly one of the floating point output formats '`%e`', '`%f`', or '`%g`'.

'`-s` *string*'
'`--separator=`*string*'
: Separate numbers with *string*; default is a newline. The output always terminates with a newline.

'-w'
'--equal-width'
Print all numbers with the same width, by padding with leading zeroes. (To have other kinds of padding, use '--format').

If you want to use `seq` to print sequences of large integer values, don't use the default '%g' format since it can result in loss of precision:

```
$ seq 1000000 1000001
1e+06
1e+06
```

Instead, you can use the format, '%1.f', to print large decimal numbers with no exponent and no decimal point.

```
$ seq --format=%1.f 1000000 1000001
1000000
1000001
```

If you want hexadecimal output, you can use `printf` to perform the conversion:

```
$ printf %x'\n' `seq -f %1.f 1048575 1024 1050623`
fffff
1003ff
1007ff
```

For very long lists of numbers, use xargs to avoid system limitations on the length of an argument list:

```
$ seq -f %1.f 1000000 | xargs printf %x'\n' | tail -n 3
f423e
f423f
f4240
```

To generate octal output, use the printf `%o` format instead of `%x`. Note however that using printf works only for numbers smaller than `2^32`:

```
$ printf "%x\n" `seq -f %1.f 4294967295 4294967296`
ffffffff
bash: printf: 4294967296: Numerical result out of range
```

On most systems, seq can produce whole-number output for values up to `2^53`, so here's a more general approach to base conversion that also happens to be more robust for such large numbers. It works by using `bc` and setting its output radix variable, *obase*, to '16' in this case to produce hexadecimal output.

```
$ (echo obase=16; seq -f %1.f 4294967295 4294967296)|bc
FFFFFFFF
100000000
```

Be careful when using `seq` with a fractional *increment*, otherwise you may see surprising results. Most people would expect to see `0.3` printed as the last number in this example:

```
$ seq -s' ' 0 .1 .3
0 0.1 0.2
```

But that doesn't happen on most systems because `seq` is implemented using binary floating point arithmetic (via the C `double` type) – which means some decimal numbers like `.1` cannot be represented exactly. That in turn means some nonintuitive conditions like `.1 * 3 > .3` will end up being true.

To work around that in the above example, use a slightly larger number as the *last* value:

```
$ seq -s' ' 0 .1 .31
0 0.1 0.2 0.3
```

In general, when using an *increment* with a fractional part, where (*last* - *first*) / *increment* is (mathematically) a whole number, specify a slightly larger (or smaller, if *increment* is negative) value for *last* to ensure that *last* is the final value printed by seq.

26 File permissions

Each file has a set of *permissions* that control the kinds of access that users have to that file. The permissions for a file are also called its *access mode*. They can be represented either in symbolic form or as an octal number.

26.1 Structure of File Permissions

There are three kinds of permissions that a user can have for a file:

1. permission to read the file. For directories, this means permission to list the contents of the directory.
2. permission to write to (change) the file. For directories, this means permission to create and remove files in the directory.
3. permission to execute the file (run it as a program). For directories, this means permission to access files in the directory.

There are three categories of users who may have different permissions to perform any of the above operations on a file:

1. the file's owner;
2. other users who are in the file's group;
3. everyone else.

Files are given an owner and group when they are created. Usually the owner is the current user and the group is the group of the directory the file is in, but this varies with the operating system, the filesystem the file is created on, and the way the file is created. You can change the owner and group of a file by using the `chown` and `chgrp` commands.

In addition to the three sets of three permissions listed above, a file's permissions have three special components, which affect only executable files (programs) and, on some systems, directories:

1. set the process's effective user ID to that of the file upon execution (called the *setuid bit*). No effect on directories.
2. set the process's effective group ID to that of the file upon execution (called the *setgid bit*). For directories on some systems, put files created in the directory into the same group as the directory, no matter what group the user who creates them is in.
3. save the program's text image on the swap device so it will load more quickly when run (called the *sticky bit*). For directories on some systems, prevent users from removing or renaming a file in a directory unless they own the file or the directory; this is called the *restricted deletion flag* for the directory.

26.2 Symbolic Modes

Symbolic modes represent changes to files' permissions as operations on single-character symbols. They allow you to modify either all or selected parts of files' permissions, optionally based on their previous values, and perhaps on the current `umask` as well (see Section 26.2.6 [Umask and Protection], page 139).

The format of symbolic modes is:

`[ugoa...][[+-=][rwxXstugo...]...][,...]`

The following sections describe the operators and other details of symbolic modes.

26.2.1 Setting Permissions

The basic symbolic operations on a file's permissions are adding, removing, and setting the permission that certain users have to read, write, and execute the file. These operations have the following format:

`users operation permissions`

The spaces between the three parts above are shown for readability only; symbolic modes cannot contain spaces.

The *users* part tells which users' access to the file is changed. It consists of one or more of the following letters (or it can be empty; see Section 26.2.6 [Umask and Protection], page 139, for a description of what happens then). When more than one of these letters is given, the order that they are in does not matter.

`u` the user who owns the file;

`g` other users who are in the file's group;

`o` all other users;

`a` all users; the same as '`ugo`'.

The *operation* part tells how to change the affected users' access to the file, and is one of the following symbols:

`+` to add the *permissions* to whatever permissions the *users* already have for the file;

`-` to remove the *permissions* from whatever permissions the *users* already have for the file;

`=` to make the *permissions* the only permissions that the *users* have for the file.

The *permissions* part tells what kind of access to the file should be changed; it is zero or more of the following letters. As with the *users* part, the order does not matter when more than one letter is given. Omitting the *permissions* part is useful only with the '`=`' operation, where it gives the specified *users* no access at all to the file.

`r` the permission the *users* have to read the file;

`w` the permission the *users* have to write to the file;

`x` the permission the *users* have to execute the file.

For example, to give everyone permission to read and write a file, but not to execute it, use:

```
a=rw
```

To remove write permission for from all users other than the file's owner, use:

```
go-w
```

The above command does not affect the access that the owner of the file has to it, nor does it affect whether other users can read or execute the file.

To give everyone except a file's owner no permission to do anything with that file, use the mode below. Other users could still remove the file, if they have write permission on the directory it is in.

```
go=
```

Another way to specify the same thing is:

```
og-rxw
```

26.2.2 Copying Existing Permissions

You can base a file's permissions on its existing permissions. To do this, instead of using '`r`', '`w`', or '`x`' after the operator, you use the letter '`u`', '`g`', or '`o`'. For example, the mode

```
o+g
```

adds the permissions for users who are in a file's group to the permissions that other users have for the file. Thus, if the file started out as mode 664 ('`rw-rw-r--`'), the above mode would change it to mode 666 ('`rw-rw-rw-`'). If the file had started out as mode 741 ('`rwxr----x`'), the above mode would change it to mode 745 ('`rwxr--r-x`'). The '`-`' and '`=`' operations work analogously.

26.2.3 Changing Special Permissions

In addition to changing a file's read, write, and execute permissions, you can change its special permissions. See Section 26.1 [Mode Structure], page 135, for a summary of these permissions.

To change a file's permission to set the user ID on execution, use '`u`' in the *users* part of the symbolic mode and '`s`' in the *permissions* part.

To change a file's permission to set the group ID on execution, use '`g`' in the *users* part of the symbolic mode and '`s`' in the *permissions* part.

To change a file's permission to stay permanently on the swap device, use '`o`' in the *users* part of the symbolic mode and '`t`' in the *permissions* part.

For example, to add set user ID permission to a program, you can use the mode:

```
u+s
```

To remove both set user ID and set group ID permission from it, you can use the mode:

```
ug-s
```

To cause a program to be saved on the swap device, you can use the mode:

```
o+t
```

Remember that the special permissions only affect files that are executable, plus, on some systems, directories (on which they have different meanings; see Section 26.1 [Mode Structure], page 135). Also, the combinations '`u+t`', '`g+t`', and '`o+s`' have no effect.

The '`=`' operator is not very useful with special permissions; for example, the mode:

```
o=t
```

does cause the file to be saved on the swap device, but it also removes all read, write, and execute permissions that users not in the file's group might have had for it.

26.2.4 Conditional Executability

There is one more special type of symbolic permission: if you use '`X`' instead of '`x`', execute permission is affected only if the file already had execute permission or is a directory. It affects directories' execute permission even if they did not initially have any execute permissions set.

For example, this mode:

```
a+X
```

gives all users permission to execute files (or search directories) if anyone could before.

26.2.5 Making Multiple Changes

The format of symbolic modes is actually more complex than described above (see Section 26.2.1 [Setting Permissions], page 136). It provides two ways to make multiple changes to files' permissions.

The first way is to specify multiple *operation* and *permissions* parts after a *users* part in the symbolic mode.

For example, the mode:

```
og+rX-w
```

gives users other than the owner of the file read permission and, if it is a directory or if someone already had execute permission to it, gives them execute permission; and it also denies them write permission to the file. It does not affect the permission that the owner of the file has for it. The above mode is equivalent to the two modes:

```
og+rX
og-w
```

The second way to make multiple changes is to specify more than one simple symbolic mode, separated by commas. For example, the mode:

```
a+r,go-w
```

gives everyone permission to read the file and removes write permission on it for all users except its owner. Another example:

```
u=rwx,g=rx,o=
```

sets all of the non-special permissions for the file explicitly. (It gives users who are not in the file's group no permission at all for it.)

The two methods can be combined. The mode:

```
a+r,g+x-w
```

gives all users permission to read the file, and gives users who are in the file's group permission to execute it, as well, but not permission to write to it. The above mode could be written in several different ways; another is:

```
u+r,g+rx,o+r,g-w
```

26.2.6 The Umask and Protection

If the *users* part of a symbolic mode is omitted, it defaults to '`a`' (affect all users), except that any permissions that are *set* in the system variable `umask` are *not affected*. The value of `umask` can be set using the `umask` command. Its default value varies from system to system.

Omitting the *users* part of a symbolic mode is generally not useful with operations other than '`+`'. It is useful with '`+`' because it allows you to use `umask` as an easily customizable protection against giving away more permission to files than you intended to.

As an example, if `umask` has the value 2, which removes write permission for users who are not in the file's group, then the mode:

```
+w
```

adds permission to write to the file to its owner and to other users who are in the file's group, but *not* to other users. In contrast, the mode:

```
a+w
```

ignores `umask`, and *does* give write permission for the file to all users.

26.3 Numeric Modes

File permissions are stored internally as integers. As an alternative to giving a symbolic mode, you can give an octal (base 8) number that corresponds to the internal representation of the new mode. This number is always interpreted in octal; you do not have to add a leading 0, as you do in C. Mode 0055 is the same as mode 55.

A numeric mode is usually shorter than the corresponding symbolic mode, but it is limited in that it cannot take into account a file's previous permissions; it can only set them absolutely.

On most systems, the permissions granted to the user, to other users in the file's group, and to other users not in the file's group are each stored as three bits, which are represented as one octal digit. The three special permissions are also each stored as one bit, and they are as a group represented as another octal digit. Here is how the bits are arranged, starting with the lowest valued bit:

```
Value in  Corresponding
Mode      Permission

          Other users not in the file's group:
   1      Execute
   2      Write
   4      Read

          Other users in the file's group:
  10      Execute
  20      Write
  40      Read
```

```
            The file's owner:
     100    Execute
     200    Write
     400    Read

            Special permissions:
    1000    Save text image on swap device
    2000    Set group ID on execution
    4000    Set user ID on execution
```

For example, numeric mode 4755 corresponds to symbolic mode '`u=rwxs,go=rx`', and numeric mode 664 corresponds to symbolic mode '`ug=rw,o=r`'. Numeric mode 0 corresponds to symbolic mode '`ugo=`'.

27 Date input formats

First, a quote:

> Our units of temporal measurement, from seconds on up to months, are so complicated, asymmetrical and disjunctive so as to make coherent mental reckoning in time all but impossible. Indeed, had some tyrannical god contrived to enslave our minds to time, to make it all but impossible for us to escape subjection to sodden routines and unpleasant surprises, he could hardly have done better than handing down our present system. It is like a set of trapezoidal building blocks, with no vertical or horizontal surfaces, like a language in which the simplest thought demands ornate constructions, useless particles and lengthy circumlocutions. Unlike the more successful patterns of language and science, which enable us to face experience boldly or at least level-headedly, our system of temporal calculation silently and persistently encourages our terror of time.
>
> . . . It is as though architects had to measure length in feet, width in meters and height in ells; as though basic instruction manuals demanded a knowledge of five different languages. It is no wonder then that we often look into our own immediate past or future, last Tuesday or a week from Sunday, with feelings of helpless confusion. . . .
>
> — Robert Grudin, *Time and the Art of Living*.

This section describes the textual date representations that GNU programs accept. These are the strings you, as a user, can supply as arguments to the various programs. The C interface (via the `getdate` function) is not described here.

Although the date syntax here can represent any possible time since the year zero, computer integers often cannot represent such a wide range of time. On POSIX systems, the clock starts at 1970-01-01 00:00:00 UTC: POSIX does not require support for times before the POSIX Epoch and times far in the future. Traditional Unix systems have 32-bit signed `time_t` and can represent times from 1901-12-13 20:45:52 through 2038-01-19 03:14:07 UTC. Systems with 64-bit signed `time_t` can represent all the times in the known lifetime of the universe.

27.1 General date syntax

A *date* is a string, possibly empty, containing many items separated by whitespace. The whitespace may be omitted when no ambiguity arises. The empty string means the beginning of today (i.e., midnight). Order of the items is immaterial. A date string may contain many flavors of items:

- calendar date items
- time of the day items
- time zone items
- day of the week items
- relative items
- pure numbers.

We describe each of these item types in turn, below.

A few numbers may be written out in words in most contexts. This is most useful for specifying day of the week items or relative items (see below). Here is the list: 'first' for 1, 'next' for 2, 'third' for 3, 'fourth' for 4, 'fifth' for 5, 'sixth' for 6, 'seventh' for 7, 'eighth' for 8, 'ninth' for 9, 'tenth' for 10, 'eleventh' for 11 and 'twelfth' for 12. Also, 'last' means exactly -1.

When a month is written this way, it is still considered to be written numerically, instead of being "spelled in full"; this changes the allowed strings.

In the current implementation, only English is supported for words and abbreviations like 'AM', 'DST', 'EST', 'first', 'January', 'Sunday', 'tomorrow', and 'year'.

The output of `date` is not always acceptable as a date string, not only because of the language problem, but also because there is no standard meaning for time zone items like 'IST'. When using `date` to generate a date string intended to be parsed later, specify a date format that is independent of language and that does not use time zone items other than 'UTC' and 'Z'. Here are some ways to do this:

```
$ LC_ALL=C TZ=UTC0 date
Fri Dec 15 19:48:05 UTC 2000
$ TZ=UTC0 date +"%Y-%m-%d %H:%M:%SZ"
2000-12-15 19:48:05Z
$ date --iso-8601=seconds  # a GNU extension
2000-12-15T11:48:05-0800
$ date --rfc-822  # a GNU extension
Fri, 15 Dec 2000 11:48:05 -0800
$ date +"%Y-%m-%d %H:%M:%S %z"  # %z is a GNU extension.
2000-12-15 11:48:05 -0800
```

Alphabetic case is completely ignored in dates. Comments may be introduced between round parentheses, as long as included parentheses are properly nested. Hyphens not followed by a digit are currently ignored. Leading zeros on numbers are ignored.

27.2 Calendar date items

A *calendar date item* specifies a day of the year. It is specified differently, depending on whether the month is specified numerically or literally. All these strings specify the same calendar date:

```
1972-09-24     # ISO 8601.
72-9-24        # Assume 19xx for 69 through 99,
               # 20xx for 00 through 68.
72-09-24       # Leading zeros are ignored.
9/24/72        # Common U.S. writing.
24 September 1972
24 Sept 72     # September has a special abbreviation.
24 Sep 72      # Three-letter abbreviations always allowed.
Sep 24, 1972
24-sep-72
24sep72
```

The year can also be omitted. In this case, the last specified year is used, or the current year if none. For example:

```
9/24
sep 24
```

Here are the rules.

For numeric months, the ISO 8601 format '*year-month-day*' is allowed, where *year* is any positive number, *month* is a number between 01 and 12, and *day* is a number between 01 and 31. A leading zero must be present if a number is less than ten. If *year* is 68 or smaller, then 2000 is added to it; otherwise, if *year* is less than 100, then 1900 is added to it. The construct '*month/day/year*', popular in the United States, is accepted. Also '*month/day*', omitting the year.

Literal months may be spelled out in full: 'January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November' or 'December'. Literal months may be abbreviated to their first three letters, possibly followed by an abbreviating dot. It is also permitted to write 'Sept' instead of 'September'.

When months are written literally, the calendar date may be given as any of the following:

```
day month year
day month
month day year
day-month-year
```

Or, omitting the year:

```
month day
```

27.3 Time of day items

A *time of day item* in date strings specifies the time on a given day. Here are some examples, all of which represent the same time:

```
20:02:0
20:02
8:02pm
20:02-0500      # In EST (U.S. Eastern Standard Time).
```

More generally, the time of the day may be given as '*hour:minute:second*', where *hour* is a number between 0 and 23, *minute* is a number between 0 and 59, and *second* is a number between 0 and 59. Alternatively, '*:second*' can be omitted, in which case it is taken to be zero.

If the time is followed by 'am' or 'pm' (or 'a.m.' or 'p.m.'), *hour* is restricted to run from 1 to 12, and '*:minute*' may be omitted (taken to be zero). 'am' indicates the first half of the day, 'pm' indicates the second half of the day. In this notation, 12 is the predecessor of 1: midnight is '12am' while noon is '12pm'. (This is the zero-oriented interpretation of '12am' and '12pm', as opposed to the old tradition derived from Latin which uses '12m' for noon and '12pm' for midnight.)

The time may alternatively be followed by a time zone correction, expressed as '*shhmm*', where *s* is '+' or '-', *hh* is a number of zone hours and *mm* is a number of zone minutes. When a time zone correction is given this way, it forces interpretation of the time relative to

Coordinated Universal Time (UTC), overriding any previous specification for the time zone or the local time zone. The *minute* part of the time of the day may not be elided when a time zone correction is used. This is the best way to specify a time zone correction by fractional parts of an hour.

Either 'am'/'pm' or a time zone correction may be specified, but not both.

27.4 Time zone items

A *time zone item* specifies an international time zone, indicated by a small set of letters, e.g., 'UTC' or 'Z' for Coordinated Universal Time. Any included periods are ignored. By following a non-daylight-saving time zone by the string 'DST' in a separate word (that is, separated by some white space), the corresponding daylight saving time zone may be specified.

Time zone items other than 'UTC' and 'Z' are obsolescent and are not recommended, because they are ambiguous; for example, 'EST' has a different meaning in Australia than in the United States. Instead, it's better to use unambiguous numeric time zone corrections like '-0500', as described in the previous section.

27.5 Day of week items

The explicit mention of a day of the week will forward the date (only if necessary) to reach that day of the week in the future.

Days of the week may be spelled out in full: 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday' or 'Saturday'. Days may be abbreviated to their first three letters, optionally followed by a period. The special abbreviations 'Tues' for 'Tuesday', 'Wednes' for 'Wednesday' and 'Thur' or 'Thurs' for 'Thursday' are also allowed.

A number may precede a day of the week item to move forward supplementary weeks. It is best used in expression like 'third monday'. In this context, 'last *day*' or 'next *day*' is also acceptable; they move one week before or after the day that *day* by itself would represent.

A comma following a day of the week item is ignored.

27.6 Relative items in date strings

Relative items adjust a date (or the current date if none) forward or backward. The effects of relative items accumulate. Here are some examples:

```
1 year
1 year ago
3 years
2 days
```

The unit of time displacement may be selected by the string 'year' or 'month' for moving by whole years or months. These are fuzzy units, as years and months are not all of equal duration. More precise units are 'fortnight' which is worth 14 days, 'week' worth 7 days, 'day' worth 24 hours, 'hour' worth 60 minutes, 'minute' or 'min' worth 60 seconds, and 'second' or 'sec' worth one second. An 's' suffix on these units is accepted and ignored.

The unit of time may be preceded by a multiplier, given as an optionally signed number. Unsigned numbers are taken as positively signed. No number at all implies 1 for a multiplier. Following a relative item by the string 'ago' is equivalent to preceding the unit by a multiplier with value -1.

The string 'tomorrow' is worth one day in the future (equivalent to 'day'), the string 'yesterday' is worth one day in the past (equivalent to 'day ago').

The strings 'now' or 'today' are relative items corresponding to zero-valued time displacement, these strings come from the fact a zero-valued time displacement represents the current time when not otherwise changed by previous items. They may be used to stress other items, like in '12:00 today'. The string 'this' also has the meaning of a zero-valued time displacement, but is preferred in date strings like 'this thursday'.

When a relative item causes the resulting date to cross a boundary where the clocks were adjusted, typically for daylight-saving time, the resulting date and time are adjusted accordingly.

27.7 Pure numbers in date strings

The precise interpretation of a pure decimal number depends on the context in the date string.

If the decimal number is of the form *yyyymmdd* and no other calendar date item (see Section 27.2 [Calendar date items], page 142) appears before it in the date string, then *yyyy* is read as the year, *mm* as the month number and *dd* as the day of the month, for the specified calendar date.

If the decimal number is of the form *hhmm* and no other time of day item appears before it in the date string, then *hh* is read as the hour of the day and *mm* as the minute of the hour, for the specified time of the day. *mm* can also be omitted.

If both a calendar date and a time of day appear to the left of a number in the date string, but no relative item, then the number overrides the year.

27.8 Authors of getdate

getdate was originally implemented by Steven M. Bellovin (smb@research.att.com) while at the University of North Carolina at Chapel Hill. The code was later tweaked by a couple of people on Usenet, then completely overhauled by Rich $alz (rsalz@bbn.com) and Jim Berets (jberets@bbn.com) in August, 1990. Various revisions for the GNU system were made by David MacKenzie, Jim Meyering, Paul Eggert and others.

This chapter was originally produced by François Pinard (pinard@iro.umontreal.ca) from the 'getdate.y' source code, and then edited by K. Berry (kb@cs.umb.edu).

28 Opening the Software Toolbox

This chapter originally appeared in *Linux Journal*, volume 1, number 2, in the *What's GNU?* column. It was written by Arnold Robbins.

Toolbox Introduction

This month's column is only peripherally related to the GNU Project, in that it describes a number of the GNU tools on your GNU/Linux system and how they might be used. What it's really about is the "Software Tools" philosophy of program development and usage.

The software tools philosophy was an important and integral concept in the initial design and development of Unix (of which Linux and GNU are essentially clones). Unfortunately, in the modern day press of Internetworking and flashy GUIs, it seems to have fallen by the wayside. This is a shame, since it provides a powerful mental model for solving many kinds of problems.

Many people carry a Swiss Army knife around in their pants pockets (or purse). A Swiss Army knife is a handy tool to have: it has several knife blades, a screwdriver, tweezers, toothpick, nail file, corkscrew, and perhaps a number of other things on it. For the everyday, small miscellaneous jobs where you need a simple, general purpose tool, it's just the thing.

On the other hand, an experienced carpenter doesn't build a house using a Swiss Army knife. Instead, he has a toolbox chock full of specialized tools—a saw, a hammer, a screwdriver, a plane, and so on. And he knows exactly when and where to use each tool; you won't catch him hammering nails with the handle of his screwdriver.

The Unix developers at Bell Labs were all professional programmers and trained computer scientists. They had found that while a one-size-fits-all program might appeal to a user because there's only one program to use, in practice such programs are

a. difficult to write,
b. difficult to maintain and debug, and
c. difficult to extend to meet new situations.

Instead, they felt that programs should be specialized tools. In short, each program "should do one thing well." No more and no less. Such programs are simpler to design, write, and get right—they only do one thing.

Furthermore, they found that with the right machinery for hooking programs together, that the whole was greater than the sum of the parts. By combining several special purpose programs, you could accomplish a specific task that none of the programs was designed for, and accomplish it much more quickly and easily than if you had to write a special purpose program. We will see some (classic) examples of this further on in the column. (An important additional point was that, if necessary, take a detour and build any software tools you may need first, if you don't already have something appropriate in the toolbox.)

I/O Redirection

Hopefully, you are familiar with the basics of I/O redirection in the shell, in particular the concepts of "standard input," "standard output," and "standard error". Briefly, "standard input" is a data source, where data comes from. A program should not need to either know or care if the data source is a disk file, a keyboard, a magnetic tape, or even a punched card reader. Similarly, "standard output" is a data sink, where data goes to. The program should neither know nor care where this might be. Programs that only read their standard input, do something to the data, and then send it on, are called *filters*, by analogy to filters in a water pipeline.

With the Unix shell, it's very easy to set up data pipelines:

```
program_to_create_data | filter1 | .... | filterN > final.pretty.data
```

We start out by creating the raw data; each filter applies some successive transformation to the data, until by the time it comes out of the pipeline, it is in the desired form.

This is fine and good for standard input and standard output. Where does the standard error come in to play? Well, think about `filter1` in the pipeline above. What happens if it encounters an error in the data it sees? If it writes an error message to standard output, it will just disappear down the pipeline into `filter2`'s input, and the user will probably never see it. So programs need a place where they can send error messages so that the user will notice them. This is standard error, and it is usually connected to your console or window, even if you have redirected standard output of your program away from your screen.

For filter programs to work together, the format of the data has to be agreed upon. The most straightforward and easiest format to use is simply lines of text. Unix data files are generally just streams of bytes, with lines delimited by the ASCII LF (Line Feed) character, conventionally called a "newline" in the Unix literature. (This is `'\n'` if you're a C programmer.) This is the format used by all the traditional filtering programs. (Many earlier operating systems had elaborate facilities and special purpose programs for managing binary data. Unix has always shied away from such things, under the philosophy that it's easiest to simply be able to view and edit your data with a text editor.)

OK, enough introduction. Let's take a look at some of the tools, and then we'll see how to hook them together in interesting ways. In the following discussion, we will only present those command line options that interest us. As you should always do, double check your system documentation for the full story.

The `who` Command

The first program is the `who` command. By itself, it generates a list of the users who are currently logged in. Although I'm writing this on a single-user system, we'll pretend that several people are logged in:

```
$ who
⊣ arnold   console Jan 22 19:57
⊣ miriam   ttyp0   Jan 23 14:19(:0.0)
⊣ bill     ttyp1   Jan 21 09:32(:0.0)
⊣ arnold   ttyp2   Jan 23 20:48(:0.0)
```

Here, the '`$`' is the usual shell prompt, at which I typed '`who`'. There are three people logged in, and I am logged in twice. On traditional Unix systems, user names are never more than eight characters long. This little bit of trivia will be useful later. The output of `who` is nice, but the data is not all that exciting.

The cut Command

The next program we'll look at is the `cut` command. This program cuts out columns or fields of input data. For example, we can tell it to print just the login name and full name from the '/etc/passwd' file. The '/etc/passwd' file has seven fields, separated by colons:

```
arnold:xyzzy:2076:10:Arnold D. Robbins:/home/arnold:/bin/bash
```

To get the first and fifth fields, we would use `cut` like this:

```
$ cut -d: -f1,5 /etc/passwd
⊣ root:Operator
...
⊣ arnold:Arnold D. Robbins
⊣ miriam:Miriam A. Robbins
...
```

With the '-c' option, `cut` will cut out specific characters (i.e., columns) in the input lines. This command looks like it might be useful for data filtering.

The sort Command

Next we'll look at the `sort` command. This is one of the most powerful commands on a Unix-style system; one that you will often find yourself using when setting up fancy data plumbing.

The `sort` command reads and sorts each file named on the command line. It then merges the sorted data and writes it to standard output. It will read standard input if no files are given on the command line (thus making it into a filter). The sort is based on the character collating sequence or based on user-supplied ordering criteria.

The uniq Command

Finally (at least for now), we'll look at the `uniq` program. When sorting data, you will often end up with duplicate lines, lines that are identical. Usually, all you need is one instance of each line. This is where `uniq` comes in. The `uniq` program reads its standard input, which it expects to be sorted. It only prints out one copy of each duplicated line. It does have several options. Later on, we'll use the '-c' option, which prints each unique line, preceded by a count of the number of times that line occurred in the input.

Putting the Tools Together

Now, let's suppose this is a large ISP server system with dozens of users logged in. The management wants the system administrator to write a program that will generate a sorted list of logged in users. Furthermore, even if a user is logged in multiple times, his or her name should only show up in the output once.

The administrator could sit down with the system documentation and write a C program that did this. It would take perhaps a couple of hundred lines of code and about two hours to write it, test it, and debug it. However, knowing the software toolbox, the administrator can instead start out by generating just a list of logged on users:

```
$ who | cut -c1-8
⊣ arnold
⊣ miriam
⊣ bill
⊣ arnold
```

Next, sort the list:

```
$ who | cut -c1-8 | sort
⊣ arnold
⊣ arnold
⊣ bill
⊣ miriam
```

Finally, run the sorted list through `uniq`, to weed out duplicates:

```
$ who | cut -c1-8 | sort | uniq
⊣ arnold
⊣ bill
⊣ miriam
```

The `sort` command actually has a '`-u`' option that does what `uniq` does. However, `uniq` has other uses for which one cannot substitute '`sort -u`'.

The administrator puts this pipeline into a shell script, and makes it available for all the users on the system ('`#`' is the system administrator, or `root`, prompt):

```
# cat > /usr/local/bin/listusers
who | cut -c1-8 | sort | uniq
^D
# chmod +x /usr/local/bin/listusers
```

There are four major points to note here. First, with just four programs, on one command line, the administrator was able to save about two hours worth of work. Furthermore, the shell pipeline is just about as efficient as the C program would be, and it is much more efficient in terms of programmer time. People time is much more expensive than computer time, and in our modern "there's never enough time to do everything" society, saving two hours of programmer time is no mean feat.

Second, it is also important to emphasize that with the *combination* of the tools, it is possible to do a special purpose job never imagined by the authors of the individual programs.

Third, it is also valuable to build up your pipeline in stages, as we did here. This allows you to view the data at each stage in the pipeline, which helps you acquire the confidence that you are indeed using these tools correctly.

Finally, by bundling the pipeline in a shell script, other users can use your command, without having to remember the fancy plumbing you set up for them. In terms of how you run them, shell scripts and compiled programs are indistinguishable.

After the previous warm-up exercise, we'll look at two additional, more complicated pipelines. For them, we need to introduce two more tools.

The first is the `tr` command, which stands for “transliterate.” The `tr` command works on a character-by-character basis, changing characters. Normally it is used for things like mapping upper case to lower case:

```
$ echo ThIs ExAmPlE HaS MIXED case! | tr '[A-Z]' '[a-z]'
⊣ this example has mixed case!
```

There are several options of interest:

`-c` work on the complement of the listed characters, i.e., operations apply to characters not in the given set

`-d` delete characters in the first set from the output

`-s` squeeze repeated characters in the output into just one character.

We will be using all three options in a moment.

The other command we'll look at is `comm`. The `comm` command takes two sorted input files as input data, and prints out the files' lines in three columns. The output columns are the data lines unique to the first file, the data lines unique to the second file, and the data lines that are common to both. The '`-1`', '`-2`', and '`-3`' command line options *omit* the respective columns. (This is non-intuitive and takes a little getting used to.) For example:

```
$ cat f1
⊣ 11111
⊣ 22222
⊣ 33333
⊣ 44444
$ cat f2
⊣ 00000
⊣ 22222
⊣ 33333
⊣ 55555
$ comm f1 f2
⊣         00000
⊣ 11111
⊣                 22222
⊣                 33333
⊣ 44444
⊣         55555
```

The single dash as a filename tells `comm` to read standard input instead of a regular file.

Now we're ready to build a fancy pipeline. The first application is a word frequency counter. This helps an author determine if he or she is over-using certain words.

The first step is to change the case of all the letters in our input file to one case. “The” and “the” are the same word when doing counting.

```
$ tr '[A-Z]' '[a-z]' < whats.gnu | ...
```

The next step is to get rid of punctuation. Quoted words and unquoted words should be treated identically; it's easiest to just get the punctuation out of the way.

```
$ tr '[A-Z]' '[a-z]' < whats.gnu | tr -cd '[A-Za-z0-9_ \012]' | ...
```

The second `tr` command operates on the complement of the listed characters, which are all the letters, the digits, the underscore, and the blank. The '`\012`' represents the newline character; it has to be left alone. (The ASCII tab character should also be included for good measure in a production script.)

At this point, we have data consisting of words separated by blank space. The words only contain alphanumeric characters (and the underscore). The next step is break the data apart so that we have one word per line. This makes the counting operation much easier, as we will see shortly.

```
$ tr '[A-Z]' '[a-z]' < whats.gnu | tr -cd '[A-Za-z0-9_ \012]' |
> tr -s '[ ]' '\012' | ...
```

This command turns blanks into newlines. The '`-s`' option squeezes multiple newline characters in the output into just one. This helps us avoid blank lines. (The '`>`' is the shell's "secondary prompt." This is what the shell prints when it notices you haven't finished typing in all of a command.)

We now have data consisting of one word per line, no punctuation, all one case. We're ready to count each word:

```
$ tr '[A-Z]' '[a-z]' < whats.gnu | tr -cd '[A-Za-z0-9_ \012]' |
> tr -s '[ ]' '\012' | sort | uniq -c | ...
```

At this point, the data might look something like this:

```
 60 a
  2 able
  6 about
  1 above
  2 accomplish
  1 acquire
  1 actually
  2 additional
```

The output is sorted by word, not by count! What we want is the most frequently used words first. Fortunately, this is easy to accomplish, with the help of two more `sort` options:

`-n` do a numeric sort, not a textual one

`-r` reverse the order of the sort

The final pipeline looks like this:

```
$ tr '[A-Z]' '[a-z]' < whats.gnu | tr -cd '[A-Za-z0-9_ \012]' |
> tr -s '[ ]' '\012' | sort | uniq -c | sort -nr
 ⊣  156 the
 ⊣   60 a
 ⊣   58 to
 ⊣   51 of
 ⊣   51 and
...
```

Whew! That's a lot to digest. Yet, the same principles apply. With six commands, on two lines (really one long one split for convenience), we've created a program that does something interesting and useful, in much less time than we could have written a C program to do the same thing.

A minor modification to the above pipeline can give us a simple spelling checker! To determine if you've spelled a word correctly, all you have to do is look it up in a dictionary. If it is not there, then chances are that your spelling is incorrect. So, we need a dictionary. The conventional location for a dictionary is '/usr/dict/words'. On my GNU/Linux system,[1] this is a is a sorted, 45,402 word dictionary.

Now, how to compare our file with the dictionary? As before, we generate a sorted list of words, one per line:

```
$ tr '[A-Z]' '[a-z]' < whats.gnu | tr -cd '[A-Za-z0-9_ \012]' |
> tr -s '[ ]' '\012' | sort -u | ...
```

Now, all we need is a list of words that are *not* in the dictionary. Here is where the `comm` command comes in.

```
$ tr '[A-Z]' '[a-z]' < whats.gnu | tr -cd '[A-Za-z0-9_ \012]' |
> tr -s '[ ]' '\012' | sort -u |
> comm -23 - /usr/dict/words
```

The '-2' and '-3' options eliminate lines that are only in the dictionary (the second file), and lines that are in both files. Lines only in the first file (standard input, our stream of words), are words that are not in the dictionary. These are likely candidates for spelling errors. This pipeline was the first cut at a production spelling checker on Unix.

There are some other tools that deserve brief mention.

`grep` search files for text that matches a regular expression

`wc` count lines, words, characters

`tee` a T-fitting for data pipes, copies data to files and to standard output

`sed` the stream editor, an advanced tool

`awk` a data manipulation language, another advanced tool

The software tools philosophy also espoused the following bit of advice: "Let someone else do the hard part." This means, take something that gives you most of what you need, and then massage it the rest of the way until it's in the form that you want.

To summarize:

1. Each program should do one thing well. No more, no less.
2. Combining programs with appropriate plumbing leads to results where the whole is greater than the sum of the parts. It also leads to novel uses of programs that the authors might never have imagined.
3. Programs should never print extraneous header or trailer data, since these could get sent on down a pipeline. (A point we didn't mention earlier.)
4. Let someone else do the hard part.
5. Know your toolbox! Use each program appropriately. If you don't have an appropriate tool, build one.

As of this writing, all the programs we've discussed are available via anonymous `ftp` from:

[1] Redhat Linux 6.1, for the November 2000 revision of this article.

`ftp://gnudist.gnu.org/textutils/textutils-1.22.tar.gz`. (There may be more recent versions available now.)

None of what I have presented in this column is new. The Software Tools philosophy was first introduced in the book *Software Tools*, by Brian Kernighan and P.J. Plauger (Addison-Wesley, ISBN 0-201-03669-X). This book showed how to write and use software tools. It was written in 1976, using a preprocessor for FORTRAN named `ratfor` (RATional FORtran). At the time, C was not as ubiquitous as it is now; FORTRAN was. The last chapter presented a `ratfor` to FORTRAN processor, written in `ratfor`. `ratfor` looks an awful lot like C; if you know C, you won't have any problem following the code.

In 1981, the book was updated and made available as *Software Tools in Pascal* (Addison-Wesley, ISBN 0-201-10342-7). The first book is still in print; the second, alas, is not. Both books are well worth reading if you're a programmer. They certainly made a major change in how I view programming.

Initially, the programs in both books were available (on 9-track tape) from Addison-Wesley. Unfortunately, this is no longer the case, although the `ratfor` versions are available from Brian Kernighan's home page (`http://cm.bell-labs.come/who/bwk`), and you might be able to find copies of the Pascal versions floating around the Internet. For a number of years, there was an active Software Tools Users Group, whose members had ported the original `ratfor` programs to essentially every computer system with a FORTRAN compiler. The popularity of the group waned in the middle 1980s as Unix began to spread beyond universities.

With the current proliferation of GNU code and other clones of Unix programs, these programs now receive little attention; modern C versions are much more efficient and do more than these programs do. Nevertheless, as exposition of good programming style, and evangelism for a still-valuable philosophy, these books are unparalleled, and I recommend them highly.

Acknowledgment: I would like to express my gratitude to Brian Kernighan of Bell Labs, the original Software Toolsmith, for reviewing this column.

Appendix A GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the

Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being list their titles, with the
Front-Cover Texts being list, and with the Back-Cover Texts being list.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being *list*"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

!

%

&

*

+

-

C

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Short Contents

Table of Contents