GNU fileutils

A set of file utilities
for version 3.16, 25 November 1996

David MacKenzie et al.

Copyright © 1994, 95, 96 Free Software Foundation, Inc.

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

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

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

1 Introduction

This manual is incomplete: No attempt is made to explain basic file 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 file utilities are mostly compatible with the POSIX.2 standard.

Please report bugs to '`fileutils-bugs@gnu.ai.mit.edu`'. 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 *GNU CC*.

This manual is based on the Unix man pages in the distribution, which were originally written by David MacKenzie and updated by Jim Meyering. François Pinard did the initial conversion to Texinfo format. Karl Berry did the indexing, some reorganization, and editing of the results. Richard Stallman contributed his usual invaluable insights to the overall process.

2 Common options

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

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

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

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`'
Make backups of files that are about to be overwritten or removed. Without this option, the original versions are destroyed.

'`-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.

'`-V` *method*'
'`--version-control=`*method*'
Use *method* to determine the type of backups made with '`-b`'. If this option is not specified, the value of the `VERSION_CONTROL` environment variable is used. And if `VERSION_CONTROL` is not set, the default backup type is '`existing`'.

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

'`t`'
'`numbered`'
Always make numbered backups.

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

'`never`'
'`simple`' Always make simple backups.

3 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.

3.1 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 files that they do not own in the directory; this is called making the directory *append-only*.

3.2 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 3.2.6 [Umask and Protection], page 7).

The format of symbolic modes is:

[ugoa...][[+-=][rwxXstugo...]...][,...]

The following sections describe the operators and other details of symbolic modes.

3.2.1 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 can not 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 3.2.6 [Umask and Protection], page 7, 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
```

3.2.2 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.

3.2.3 Changing Special Permissions

In addition to changing a file's read, write, and execute permissions, you can change its special permissions. See Section 3.1 [Mode Structure], page 3, 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 3.1 [Mode Structure], page 3). Using '`a`' in the *users* part of a symbolic mode does not cause the special permissions to be affected; thus,

```
a+s
```

has *no effect*. You must use '`u`', '`g`', and '`o`' explicitly to affect the special permissions. 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.

3.2.4 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.

3.2.5 Making Multiple Changes

The format of symbolic modes is actually more complex than described above (see Section 3.2.1 [Setting Permissions], page 4). 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 it 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
```

3.2.6 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.

3.3 Numeric Modes

File permissions are stored internally as 16 bit 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 can not take into account a file's previous permissions; it can only set them absolutely.

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 in the 16 bit integer, 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=`'.

4 Date input formats

> 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 zero A.D., computer integers are not big enough for such a (comparatively) long time. The earliest date semantically allowed on Unix systems is midnight, 1 January 1970 UCT.

4.1 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.

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.

4.2 Calendar date item

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:

```
1970-09-17          # ISO 8601.
70-9-17             # This century assumed by default.
70-09-17            # Leading zeros are ignored.
9/17/72             # 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:

```
9/17
sep 17
```

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 less than 100, then 1900 is added to it to force a date in this century. 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

4.3 Time of day item

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 (Eastern U.S. 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`'.

The time may alternatively be followed by a timezone 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 timezone correction is given this way, it forces interpretation of the time in UTC, overriding any previous specification for the timezone or the local timezone. The *minute* part of the time of the day may not be elided when a timezone correction is used. This is the only way to specify a timezone correction by fractional parts of an hour.

Either '`am`'/'`pm`' or a timezone correction may be specified, but not both.

4.4 Timezone item

A *timezone item* specifies an international timezone, indicated by a small set of letters. Any included period is ignored. Military timezone designations use a single letter. Currently, only integral zone hours may be represented in a timezone item. See the previous section for a finer control over the timezone correction.

Here are many non-daylight-savings-time timezones, indexed by the zone hour value.

+000 '`GMT`' for Greenwich Mean, '`UT`' or '`UTC`' for Universal (Coordinated), '`WET`' for Western European and '`Z`' for militaries.

+100 '`WAT`' for West Africa and '`A`' for militaries.

+200 '`AT`' for Azores and '`B`' for militaries.

+300 '`C`' for militaries.

+400 '`AST`' for Atlantic Standard and '`D`' for militaries.

+500 '`E`' for militaries and '`EST`' for Eastern Standard.

+600 '`CST`' for Central Standard and '`F`' for militaries.

+700 '`G`' for militaries and '`MST`' for Mountain Standard.

+800 '`H`' for militaries and '`PST`' for Pacific Standard.

+900 'I' for militaries and 'YST' for Yukon Standard.

+1000 'AHST' for Alaska-Hawaii Standard, 'CAT' for Central Alaska, 'HST' for Hawaii Standard and 'K' for militaries.

+1100 'L' for militaries and 'NT' for Nome.

+1200 'IDLW' for International Date Line West and 'M' for militaries.

-100 'CET' for Central European, 'FWT' for French Winter, 'MET' for Middle European, 'MEWT' for Middle European Winter, 'N' for militaries and 'SWT' for Swedish Winter.

-200 'EET' for Eastern European, USSR Zone 1 and 'O' for militaries.

-300 'BT' for Baghdad, USSR Zone 2 and 'P' for militaries.

-400 'Q' for militaries and 'ZP4' for USSR Zone 3.

-500 'R' for militaries and 'ZP5' for USSR Zone 4.

-600 'S' for militaries and 'ZP6' for USSR Zone 5.

-700 'T' for militaries and 'WAST' for West Australian Standard.

-800 'CCT' for China Coast, USSR Zone 7 and 'U' for militaries.

-900 'JST' for Japan Standard, USSR Zone 8 and 'V' for militaries.

-1000 'EAST' for East Australian Standard, 'GST' for Guam Standard, USSR Zone 9 and 'W' for militaries.

-1100 'X' for militaries.

-1200 'IDLE' for International Date Line East, 'NZST' for New Zealand Standard, 'NZT' for New Zealand and 'Y' for militaries.

Here are many DST timezones, indexed by the zone hour value. Also, by following a non-DST timezone by the string 'DST' in a separate word (that is, separated by some whitespace), the corresponding DST timezone may be specified.

0 'BST' for British Summer.

+400 'ADT' for Atlantic Daylight.

+500 'EDT' for Eastern Daylight.

+600 'CDT' for Central Daylight.

+700 'MDT' for Mountain Daylight.

+800 'PDT' for Pacific Daylight.

+900 'YDT' for Yukon Daylight.

+1000 'HDT' for Hawaii Daylight.

-100 'MEST' for Middle European Summer, 'MESZ' for Middle European Summer, 'SST' for Swedish Summer and 'FST' for French Summer.

-700 'WADT' for West Australian Daylight.

-1000 'EADT' for Eastern Australian Daylight.

-1200 'NZDT' for New Zealand Daylight.

4.5 Day of week item

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.

4.6 Relative item 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 multiplicator 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 change 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 makes the resulting date to cross the boundary between DST and non-DST (or vice-versa), the hour is adjusted according to the local time.

4.7 Pure numbers in date strings

The precise intepretation of a pure decimal number is dependent of the context in the date string.

If the decimal number is of the form *yyyymmdd* and no other calendar date item (see Section 4.2 [Calendar date item], page 10) 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.

4.8 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, 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`’).

5 Directory listing

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

5.1 `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 arguments are specified, `ls` lists the contents of the current directory.

By default, the output is sorted alphabetically. If standard output is a terminal, the output is in columns (sorted vertically); otherwise, they are listed one per line.

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.

The '`-g`' option is accepted but ignored, for compatibility with Unix. Also see Chapter 2 [Common options], page 2.

5.1.1 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.

'`-I`'
'`--ignore`'
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*.

'-L'
'--dereference'
: In a long listing, show file information (e.g., times and permissions) for the referents of symbolic links rather than for the symbolic links themselves.

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

5.1.2 What information is listed

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

'-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 which 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 after each subdirectory:

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

'-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.)

'-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 in bytes, and timestamp (by default, the modification time). For files with a time more than six months old or more than one hour into the future, the timestamp contains the year instead of the time of day.

  For each directory that is listed, preface the files with a line '`total` *blocks*', where *blocks* is the total disk space used by all files in that directory. By default, 1024-byte blocks are used; if the environment variable `POSIXLY_CORRECT` is set,

512-byte blocks are used (unless the '`-k`' option is given). The *blocks* computed counts each hard link separately; this is arguably a deficiency.

The permissions listed are similar to symbolic mode specifications (see Section 3.2 [Symbolic Modes], page 3). 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.

'`-o`' Produce long format directory listings, but don't display group information. It is equivalent to using '`--format=long`' with '`--no-group`' . This option is provided for compatibility with other versions of `ls`.

'`-s`'
'`--size`' Print the size of each file in 1024-byte blocks to the left of the file name. If the environment variable `POSIXLY_CORRECT` is set, 512-byte blocks are used instead, unless the '`-k`' option is given (see Section 5.1.4 [General output formatting], page 18).

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.

5.1.3 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`'
Sort according to the status change time (the '`ctime`' in the inode). If the long listing format ('`-l`') is being used, print the status change time instead of the modification 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`'
'`--time=use`'
: Sort by access time (the '`atime`' in the inode). If the long listing format is being used, print the last 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.

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

5.1.4 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.

'`--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`'.

'`-F`'
'`--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.

'`--full-time`'
: List times in full, rather than using the standard abbreviation heuristics. The format is the same as `date`'s default; it's not possible to change this, but you can extract out the date string with `cut` and then pass the result to `date -d`. See section "date invocation" in *Shell utilities*.

  This is most useful because the time output includes the seconds. (Unix filesystems store file timestamps only to the nearest second, so this option shows all the information there is.) For example, this can help when you have a Makefile that is not regenerating files properly.

'`-k`'
'`--kilobytes`'
: If file sizes are being listed, print them in kilobytes. This overrides the environment variable `POSIXLY_CORRECT`.

'`-m`'
'`--format=commas`'
: List files horizontally, with as many as will fit on each line, separated by '`, `' (a comma and a space).

'`-n`'
'`--numeric-uid-gid`'
: List the numeric UID and GID instead of the names.

'`-p`' 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.

5.1.5 Formatting the file names

These options change how file names themselves are printed.

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

'`-N`'
'`--literal`'
: Do not quote file names.

'`-q`'
'`--hide-control-chars`'
: Print question marks instead of nongraphic characters in file names. This is the default.

'`-Q`'
'`--quote-name`'
: Enclose file names in double quotes and quote nongraphic characters as in C.

5.2 `dir`: Briefly list directory contents

`dir` (also installed as `d`) is equivalent to `ls -C`; that is, files are by default listed in columns, sorted vertically.

See Section 5.1 [ls invocation], page 15.

5.3 `vdir`: Verbosely list directory contents

`vdir` (also installed as `v`)is equivalent to `ls -l`; that is, files are by default listed in long format.

5.4 `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.

6 Basic operations

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

6.1 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 (see '`-r`' below).

`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 preserve directory structure). Equivalent to '`-dpR`'.

'`-b`'
'`--backup`'
: Make backups of files that are about to be overwritten or removed. See Section 2.1 [Backup options], page 2.

'`-d`'
'`--no-dereference`'
: 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.

'`-f`'
'`--force`' Remove existing destination files.

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

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

'`-p`'
'`--preserve`'
Preserve the original files' owner, group, permissions, and timestamps.

'`-p`'
'`--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.

'`-r`' Copy directories recursively, copying any non-directories and non-symbolic links (that is, FIFOs and special files) as if they were regular files. This means trying to read the data in each source file and writing it to the destination. Thus, with this option, `cp` may well hang indefinitely reading a FIFO, unless something else happens to be writing it.

'`-R`'
'`--recursive`'
Copy directories recursively, preserving non-directories (see '`-r`' just 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. If you find an application for this option, let us know.

'`-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.

'`-u`'
'`--update`'
: Do not copy a nondirectory that has an existing destination with the same or newer modification time.

'`-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 '`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.

6.2 `dd`: Convert and copy a file

`dd` copies a file (from standard input to standard output, by default) with a changeable I/O blocksize, 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, '`k`'=1024, '`w`'=2, '`x`*m*'=*m*.

'`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* '`obs`'-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.

6.3 `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`'
: Make backups of files that are about to be overwritten or removed. See Section 2.1 [Backup options], page 2.

'`-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's 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 3 [File permissions], page 3). The default mode is 0755—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.

'`-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.

'`-V` *method*'
'`--version-control=`*method*'
: Change the type of backups made with '`-b`'. The *method* argument can be '`numbered`' (or '`t`'), '`existing`' (or '`nil`'), or '`never`' (or '`simple`'). See Section 2.1 [Backup options], page 2.

6.4 `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 only regular files across filesystems.

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.

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

'`-b`'
'`--backup`'
: Make backups of files that are about to be overwritten or removed. See Section 2.1 [Backup options], page 2.

'`-f`'
'`--force`'
: Remove existing destination files and never prompt the user.

'`-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.

'`-u`'
'`--update`'
: Do not move a nondirectory that has an existing destination with the same or newer modification time.

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

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

'`-V` *method*'
'`--version-control=`*method*'
: Change the type of backups made with '`-b`'. The *method* argument can be '`numbered`' (or '`t`'), '`existing`' (or '`nil`'), or '`never`' (or '`simple`'). See Section 2.1 [Backup options], page 2.

6.5 `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.

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

'`-d`'
'`--directory`'
: Remove directories with `unlink` instead of `rmdir`, and don't require a directory to be empty before trying to unlink it. Only works if you have appropriate privileges. 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.

'`-i`'
'`--interactive`'
: Prompt whether to remove each file. If the response does not begin with '`y`' or '`Y`', the file is skipped.

'`-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 being 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.

7 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*.

7.1 `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]... source [dest]
ln [option]... source... directory
```

If the last argument names an existing directory, `ln` links each *source* file into a file with the same name in that directory. (But see the description of the '`--no-dereference`' option below.) If only one file is given, it links that file into the current directory. Otherwise, if only two files are given, it links the first onto the second. It is an error if the last argument is not a directory and more than two files are given. By default, it does not remove 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 links to directories, 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; in particular, 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 *GNU C library*.

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

'`-b`'
'`--backup`'
: Make backups of files that are about to be overwritten or removed. See Section 2.1 [Backup options], page 2.

'`-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.

'`-v`'
'`--verbose`'
Print the name of each file before linking it.

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

'`-V` *method*'
'`--version-control=`*method*'
Change the type of backups made with '`-b`'. The *method* argument can be '`numbered`' (or '`t`'), '`existing`' (or '`nil`'), or '`never`' (or '`simple`'). See Section 2.1 [Backup options], page 2.

7.2 `mkdir`: Make directories

`mkdir` creates directories with the specified names. Synopsis:

```
mkdir [option]... name...
```

It is not an error if a *name* is already a directory; `mkdir` simply proceeds. But if a *name* is an existing file and is anything but a directory, `mkdir` complains.

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 0777 (read, write and execute allowed for everyone) minus the bits set in the umask for the point of the departure. See Chapter 3 [File permissions], page 3.

'`-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.

'`--verbose`'
: Print a message for each created directory. This is most useful with '`--parents`'.

7.3 `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 0666 (read and write allowed for everyone) minus the bits set in the umask for the point of departure. See Chapter 3 [File permissions], page 3.

7.4 `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 (buffered) special file

'`c`' for a character (buffered) special file

'`u`' for a character (unbuffered) 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 0666 minus the bits set in the umask as the point of departure. See Chapter 3 [File permissions], page 3.

7.5 `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.

'`-p`'
'`--parents`'
: Remove any parent directories that become empty after an argument *directory* is removed.

See Section 6.5 [rm invocation], page 28, for how to remove non-empty directories (recursively).

8 Changing file attributes

A file is not merely its contents, a name, and a file type (see Chapter 7 [Special file types], page 29). 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.

8.1 `chown`: Change file owner and group

`chown` changes the user and/or group ownership of each given file. Synopsis:

```
chown [option]... new-owner file...
```

The first non-option argument, *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*
owner':'*group*
: If the *owner* is followed by a colon or dot 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'.'
owner':' If a colon or dot 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*
':'*group* If the colon or dot 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`.

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.

'`-h`'
'`--no-dereference`'
: Act on symbolic links themselves instead of what they point to. Only available if the `lchown` system call is provided.

'`-v`'
'`--verbose`'
: Verbosely describe the action (or non-action) taken for every *file*.

'`-R`'
'`--recursive`'
: Recursively change ownership of directories and their contents.

8.2 `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. Synopsis:

```
chgrp [option]... group 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.

'`-f`'
'`--no-dereference`'
: Act on symbolic links themselves instead of what they point to. Only available if the `lchown` system call is provided.

'`-v`'
'`--verbose`'
: Verbosely describe the action or non-action taken for every *file*.

'`-R`'
'`--recursive`'
: Recursively change the group ownership of directories and their contents.

8.3 `chmod`: Change access permissions

`chmod` changes the access permissions of the named files. Synopsis:

```
chmod [option]... mode 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.

The first non-option argument, *mode*, specifies the new permissions. See the section below for details.

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*.

'`-R`'
'`--recursive`'
: Recursively change permissions of directories and their contents.

8.4 `touch`: Change file timestamps

`touch` changes the access and/or modification times of the specified files. Synopsis:

```
touch [option]... file...
```

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.

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.

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, timezones, '`am`' and '`pm`', etc. See Chapter 4 [Date input formats], page 9.

'`-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 MMDDhhmm[[CC]YY][.ss]`’
: Use the argument (months, days, hours, minutes, optional century and years, optional seconds) instead of the current time.

9 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.)

9.1 `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*.

Disk space is shown in 1024-byte blocks by default, unless the environment variable `POSIXLY_CORRECT` is set, in which case 512-byte blocks are used (unless the '`-k`' option is given).

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 nonportable 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.

'`-h`'
'`--human-readable`'
Append a size letter such as '`M`' for megabytes to each size.

'`-i`'
'`--inodes`'
List inode usage information instead of block usage. An inode (short for index node) is contains information about a file such as its owner, permissions, timestamps, and location on the disk.

'`-k`'
'`--kilobytes`'
Print sizes in 1024-byte blocks. This overrides the environment variable `POSIXLY_CORRECT`.

'`-m`'
'`--megabytes`'
Print sizes in megabyte (that 1,048,576 bytes) blocks.

'`--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 that 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.

'`--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`.

9.2 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. The output is in 1024-byte units by default, unless the environment variable `POSIXLY_CORRECT` is set, in which case 512-byte blocks are used (unless '`-k`' is specified).

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, instead of kilobytes.

'`-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, such as '`M`' for megabytes, to each size.

'`-k`'
'`--kilobytes`'
: Print sizes in kilobytes. This overrides the environment variable `POSIXLY_CORRECT`.

'`-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).

'`-m`'
'`--megabytes`'
: Print sizes in megabyte (that 1,048,576 bytes) blocks.

'`-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.

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.

9.3 `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).

Index

-

- and Unix `rm` ... 28
'-', removing files beginning with ... 28
`--all` ... 15, 37, 39
`--almost-all` ... 15
`--archive` ... 22
`--backup` ... 2, 22, 26, 27, 29
`--bourne-shell` ... 21
`--bytes` ... 39
`--c-shell` ... 21
`--changes` ... 33, 34, 35
`--classify` ... 19
`--color` ... 18
`--count-links` ... 39
`--csh` ... 21
`--date` ... 35
`--dereference` ... 16, 40
`--dereference-args` ... 39
`--directory` ... 15, 26, 28, 30
`--dired` ... 16
`--escape` ... 20
`--exclude-type` ... 39
`--force` ... 22, 27, 28, 30
`--format` ... 16, 18, 19
`--full-time` ... 19
`--group` ... 26
`--help` ... 2
`--hide-control-chars` ... 20
`--human-readable` ... 37, 39
`--ignore-backups` ... 15
`--ignore=`*pattern* ... 15
`--inode` ... 16
`--inodes` ... 37
`--interactive` ... 22, 27, 28, 30
`--kilobytes` ... 19, 37, 39
`--link` ... 23
`--literal` ... 20
`--megabytes` ... 38, 40
`--mode` ... 26, 31, 32
`--no-create` ... 35
`--no-dereference` ... 22, 30, 33, 34
`--no-group` ... 16
`--no-sync` ... 38
`--numeric-uid-gid` ... 19
`--one-file-system` ... 24, 40
`--owner` ... 26
`--parents` ... 23, 31, 32
`--portability` ... 38
`--preserve` ... 23
`--print-database` ... 21
`--print-type` ... 38
`--quiet` ... 33, 34, 35
`--quote-name` ... 20
`--recursive` ... 16, 23, 28, 34, 35
`--reference` ... 36
`--reverse` ... 18
`--separate-dirs` ... 40
`--sh` ... 21
`--silent` ... 33, 34, 35
`--size` ... 17
`--sort` ... 18
`--sparse=`*when* ... 23
`--strip` ... 26
`--suffix` ... 2, 24, 26, 27, 30
`--summarize` ... 40
`--symbolic` ... 30
`--symbolic-link` ... 23
`--sync` ... 38
`--tabsize` ... 19
`--time` ... 17, 18, 35, 36
`--total` ... 39
`--type` ... 38
`--update` ... 24, 27
`--verbose` ... 24, 27, 28, 30, 31, 34, 35
`--version` ... 2
`--version-control` ... 2, 24, 26, 27, 30
`--width` ... 20
`-1` ... 18
`-a` ... 15, 22, 35, 37, 39
`-A` ... 15
`-b` ... 2, 20, 21, 22, 26, 27, 29, 39
`-B` ... 15
`-c` ... 17, 21, 26, 33, 34, 35, 39
`-C` ... 18
`-d` ... 15, 22, 26, 28, 30, 35
`-D` ... 16, 39
`-f` ... 17, 22, 27, 28, 30, 33, 34, 35
`-F` ... 19, 30
`-g` ... 26
`-G` ... 16
`-g` (ignored) ... 15
`-h` ... 33, 37, 39
`-i` ... 16, 22, 27, 28, 30, 37
`-I` ... 15

-k ... 19, 37, 39
-l ... 16, 23, 39
-L ... 16, 40
-m ... 19, 26, 31, 32, 36, 38, 40
-n ... 19, 30
-N ... 20
-o ... 17, 26
-p ... 21, 23, 31, 32
-P ... 23, 38
-q ... 20
-Q ... 20
-r ... 18, 28, 36
-R ... 16, 23, 28, 34, 35
-s ... 17, 23, 26, 30, 40
-S ... 2, 18, 24, 26, 27, 30, 40
-t ... 18, 38
-T ... 19, 38
-u ... 18, 24, 27
-U ... 18
-v ... 24, 27, 28, 30, 34, 35
-V ... 2, 24, 26, 27, 30
-w ... 20
-x ... 19, 24, 39, 40
-X ... 18

4

4.2 filesystem type ... 38

A

abbreviations for months ... 10
access permissions, changing ... 34
access time, changing ... 35
access time, sorting files by ... 18
across, listing files ... 19
adding permissions ... 4
ago in date strings ... 13
Alaska-Hawaii Time ... 12
alternate ebcdic, converting to ... 25
always color option ... 19
am in date strings ... 11
append-only directories ... 3
appropriate privileges ... 26
ascii, converting to ... 25
atime, changing ... 35
atime, sorting files by ... 18
Atlantic Standard Time ... 11
attributes, file ... 33
authors of getdate ... 14
auto color option ... 18
automounter filesystems ... 37
Azores Time ... 11

B

b for block special file ... 32
backslash sequences for file names ... 20
backup files, ignoring ... 15
backup files, type made ... 2
backup options ... 2
backup suffix ... 2
backups, making ... 2, 22, 26, 27, 29
backups, making only ... 22
Baghdad Time ... 12
beginning of time, for Unix ... 9
Bellovin, Steven M. ... 14
Berets, Jim ... 14
Berry, K. ... 14
block (space-padding) ... 25
block size ... 24
block size of conversion ... 24
block size of input ... 24
block size of output ... 24
block special files ... 31
block special files, creating ... 31
Bourne shell syntax for color setup ... 21
bs ... 24
BSD touch compatibility ... 35
buffered character file ... 32
bugs, reporting ... 1
byte-swapping ... 25

C

c for character special file ... 32
C shell syntax for color setup ... 21
calendar date item ... 10
case, ignored in dates ... 10
cbs ... 24
CD-ROM filesystem type ... 38
cdfs filesystem type ... 38
Central Alaska Time ... 12
Central European Time ... 12
Central Standard Time ... 11
changed files, verbosely describing ... 34
changed owners, verbosely describing ... 33
changing access permissions ... 34
changing file attributes ... 33
changing file ownership ... 33
changing file timestamps ... 35
changing group ownership ... 33, 34
changing special permissions ... 5
character special files ... 31

character special files, creating ... 31
`chgrp` ... 34
China Coast Time ... 12
`chmod` ... 34
`chown` ... 33
color database, printing ... 21
color setup ... 20
color, distinguishing file types with ... 18
`COLUMNS` ... 20
`commas`, outputting between files ... 19
comments, in dates ... 10
common options ... 2
conditional executability ... 6
`conv` ... 25
converstion block size ... 24
converting while copying a file ... 24
copying directories recursively ... 23
copying existing permissions ... 5
copying files and directories ... 22
copying files and setting attributes ... 25
`count` ... 25
`cp` ... 22
crashes and corruption ... 40
creating directories ... 30
creating FIFOs (named pipes) ... 31
creating links (hard or soft) ... 29
`csh` syntax for color setup ... 21
`ctime`, sorting by ... 17

D

database for color setup, printing ... 21
date format, ISO 8601 ... 10
date input formats ... 9
`day` in date strings ... 13
day of week item ... 13
daylight savings time ... 12
`dd` ... 24
dereferencing symbolic links ... 29
device file, disk ... 37
`df` ... 37
`dir` ... 20
`dircolors` ... 20
directories, copying ... 22
directories, copying recursively ... 23
directories, creating ... 30
directories, creating with given attributes ... 26
directories, removing (recursively) ... 28
directories, removing empty ... 32
directories, removing with `unlink` ... 28
directory listing ... 15
directory listing, brief ... 20
directory listing, recursive ... 16
directory listing, verbose ... 20
directory order, listing by ... 17
dired Emacs mode support ... 16
disk device file ... 37
disk usage ... 37
disk usage by filesystem ... 37
disk usage for files ... 39
diskette filesystem ... 38
displacement of dates ... 13
DOS filesystem ... 38
`du` ... 39

E

East Australian Standard Time ... 12
Eastern European Time ... 12
Eastern Standard Time ... 11
`ebcdic`, converting to ... 25
`efs` filesystem type ... 38
empty files, creating ... 35
epoch, for Unix ... 9
error messages, omitting ... 33, 34, 35
executables and file type, marking ... 19
execute permission ... 3
execute permission, symbolic ... 4
`existing` backup method ... 2
`extension`, sorting files by ... 18

F

FIFOs, creating ... 31
file attributes, changing ... 33
file information, preserving ... 23
file ownership, changing ... 33
file permissions ... 3
file permissions, numeric ... 7
file space usage ... 39
file timestamps, changing ... 35
file type and executables, marking ... 19
file type, marking ... 19
file types ... 29
file types, special ... 29
files beginning with '`-`', removing ... 28
files, copying ... 22
filesystem disk usage ... 37
filesystem space, retrieving current data more slowly ... 38
filesystem space, retrieving old data more quickly ... 38
filesystem types, limiting output to certain ... 38

filesystem types, printing 38
filesystems and hard links 29
filesystems, omitting copying to different 24
`first` in date strings 9
`fortnight` in date strings 13
French Winter Time 12
`fsck` 28

G

general date syntax 9
`getdate` 9
giving away permissions 7
grand total of disk space 39
Greenwich Mean Time 11
group owner, default 3
group ownership of installed files, setting 26
group ownership, changing 34
group ownerships, changing 33
group, permissions for 4
Guam Standard Time 12

H

hard links to directories 30
hard links, counting in `du` 39
hard links, creating 29
hard links, preserving 22
Hawaii Standard Time 12
help, online 2
High Sierra filesystem 38
history 1
holes, copying files with 23
`horizontal`, listing files 19
`hour` in date strings 13
`hsfs` filesystem type 38
human-readable `df` output 37, 39

I

`ibs` 24
`if` 24
ignore filesystems 37
inode number, printing 16
inode usage 37
inodes, written buffered 40
input block size 24
`install` 25
International Date Line East 12
International Date Line West 12
introduction 1
ISO 8601 date format 10
items in date strings 9

J

Japan Standard Time 12

K

kilobytes for filesystem sizes 37

L

`last` *day* 13
`last` in date strings 9
`lcase`, converting to 25
`lchown` 33, 34
leading directories, creating missing 26
links, creating 29
Linux filesystem types 38
`ln` 29
local filesystem types 38
`long ls` format 16
`ls` 15
`LS_COLORS` 20

M

MacKenzie, David 14
Makefiles, installing programs in 25
manipulating files 22
megabytes for filesystem sizes 38, 40
Meyering, Jim 14
Middle European Time 12
Middle European Winter Time 12
`midnight` in date strings 11
`minute` in date strings 13
minutes, timezone correction by 11
`mkdir` 30
`mkfifo` 31
`mknod` 31
modes and umask 7
modes of created directories, setting 31
modes of created FIFOs, setting 31
`modification time`, sorting files by 18
`modify` time, changing 36
`month` in date strings 13
month names in date strings 10
months, written-out 10
Mountain Standard Time 11
MS-DOS filesystem 38
`mtime`, changing 36
multiple changes to permissions 6
multipliers after numbers 24
`mv` 27

N

named pipes, creating ... 31
New Zealand Standard Time ... 12
newer files, copying only ... 24
newer files, moving only ... 27
`next` *day* ... 13
`next` in date strings ... 9
NFS filesystem type ... 38
NFS mounts from BSD to HP-UX ... 17, 40
`noerror` ... 25
Nome Standard Time ... 12
non-directories, copying as special files ... 23
`none` color option ... 18
`none`, sorting option for `ls` ... 18
`noon` in date strings ... 11
`notrunc` ... 25
`now` in date strings ... 13
`numbered` backup method ... 2
numbers, written-out ... 9
numeric modes ... 7
numeric uid and gid ... 19

O

`obs` ... 24
octal numbers for file modes ... 7
`of` ... 24
one filesystem, restricting `du` to ... 40
one-line output format ... 38
ordinal numbers ... 9
other permissions ... 4
output block size ... 24
output format, portable ... 38
owner of file, permissions for ... 4
owner, default ... 3
ownership of installed files, setting ... 26

P

`p` for FIFO file ... 32
Pacific Standard Time ... 11
parent directories and `cp` ... 23
parent directories, creating ... 31
parent directories, creating missing ... 26
parent directories, removing ... 32
PC filesystem ... 38
`pcfs` ... 38
permissions of files ... 3
permissions of installed files, setting ... 26
permissions, changing access ... 34
permissions, copying existing ... 5
permissions, for changing file timestamps ... 35
permissions, output by `ls` ... 17
Pinard, F. ... 14
`pm` in date strings ... 11
portable output format ... 38
POSIX output format ... 38
POSIX.2 ... 1
`POSIXLY_CORRECT` ... 37
`POSIXLY_CORRECT`, overridden by `df -k` ... 37
`POSIXLY_CORRECT`, overridden by `du -k` ... 39
`POSIXLY_CORRECT`, overridden by `ls -k` ... 19
`POSIXLY_CORRECT`, overrides `ls -s` ... 17
printing color database ... 21
prompting, and `ln` ... 30
prompting, and `mv` ... 27
prompting, and `rm` ... 28
prompts, forcing ... 27
prompts, omitting ... 27
pure numbers in date strings ... 13

R

read errors, ignoring ... 25
read permission ... 3
read permission, symbolic ... 4
`read` system call, and holes ... 23
recursive directory listing ... 16
recursively changing access permissions ... 35
recursively changing file ownership ... 34
recursively changing group ownership ... 34
recursively copying directories ... 23
relative items in date strings ... 13
removing empty directories ... 32
removing files or directories ... 28
removing permissions ... 4
reverse sorting ... 18
`rm` ... 28
`rmdir` ... 32
`root` as default owner ... 26

S

Salz, Rich ... 14
`seek` ... 25
self-backups ... 22
setgid ... 3
setting permissions ... 4
setuid ... 3
setup for color ... 20
`sh` syntax for color setup ... 21
`SHELL` environment variable, and color ... 20
`simple` backup method ... 2
`SIMPLE_BACKUP_SUFFIX` ... 2

`single-column` output of files ... 18
size of files, reporting ... 17
`size of files`, sorting files by ... 18
`skip` ... 25
sorting `ls` output ... 17
sparse files, copying ... 23
special file types ... 29
special files ... 31
`status time`, sorting by ... 17
sticky ... 3
stripping symbol table information ... 26
subtracting permissions ... 4
superblock, writing ... 40
`swab` (byte-swapping) ... 25
swap space, saving text image in ... 3
Swedish Winter Time ... 12
symbol table information, stripping ... 26
symbolic (soft) links, creating ... 29
symbolic links, changing group ... 34
symbolic links, changing owner ... 33
symbolic links, copying ... 22
symbolic links, copying with ... 23
symbolic links, dereferencing ... 16
symbolic links, permissions of ... 34
symbolic modes ... 3
`sync` ... 40
`sync` (padding with nulls) ... 25
synchronize disk and memory ... 40

T

terminal, using color iff ... 18
text image, saving in swap space ... 3
`this` in date strings ... 13
`time` ... 35
time of day item ... 11
timestamps, changing file ... 35
timezone correction ... 11
timezone item ... 11
`today` in date strings ... 13
`tomorrow` in date strings ... 13
`touch` ... 35
truncating output file, avoiding ... 25

U

`u` for unbuffered character special file ... 32
`ucase`, converting to ... 25
`ufs` filesystem type ... 38
umask and modes ... 7
`unblock` ... 25
unbuffered character special file ... 32
Universal Coordinated Time ... 11
`unlink` ... 28
unsorted directory listing ... 17
`use` time, changing ... 35
`use time`, sorting files by ... 18
USSR Zone ... 12

V

`vdir` ... 20
`verbose ls` format ... 16
version number, finding ... 2
`version-control` Emacs variable ... 2
`VERSION_CONTROL` ... 2
`vertical` sorted files in columns ... 18

W

`week` in date strings ... 13
West African Time ... 11
West Australian Standard Time ... 12
Western European Time ... 11
write permission ... 3
write permission, symbolic ... 4

Y

`year` in date strings ... 13
`yesterday` in date strings ... 13
Yukon Standard Time ... 12

Table of Contents

1 Introduction 1

2 Common options 2
- 2.1 Backup options 2

3 File permissions 3
- 3.1 Structure of File Permissions 3
- 3.2 Symbolic Modes 3
  - 3.2.1 Setting Permissions 4
  - 3.2.2 Copying Existing Permissions 5
  - 3.2.3 Changing Special Permissions 5
  - 3.2.4 Conditional Executability 6
  - 3.2.5 Making Multiple Changes 6
  - 3.2.6 The Umask and Protection 7
- 3.3 Numeric Modes 7

4 Date input formats 9
- 4.1 General date syntax 9
- 4.2 Calendar date item 10
- 4.3 Time of day item 11
- 4.4 Timezone item 11
- 4.5 Day of week item 13
- 4.6 Relative item in date strings 13
- 4.7 Pure numbers in date strings 13
- 4.8 Authors of `getdate` 14

5 Directory listing 15
- 5.1 `ls`: List directory contents 15
  - 5.1.1 Which files are listed 15
  - 5.1.2 What information is listed 16
  - 5.1.3 Sorting the output 17
  - 5.1.4 General output formatting 18
  - 5.1.5 Formatting the file names 20
- 5.2 `dir`: Briefly list directory contents 20
- 5.3 `vdir`: Verbosely list directory contents 20
- 5.4 `dircolors`: Color setup for `ls` 20

6 Basic operations 22

6.1 `cp`: Copy files and directories 22
6.2 `dd`: Convert and copy a file 24
6.3 `install`: Copy files and set attributes 25
6.4 `mv`: Move (rename) files 27
6.5 `rm`: Remove files or directories 28

7 Special file types 29

7.1 `ln`: Make links between files 29
7.2 `mkdir`: Make directories 30
7.3 `mkfifo`: Make FIFOs (named pipes) 31
7.4 `mknod`: Make block or character special files 31
7.5 `rmdir`: Remove empty directories 32

8 Changing file attributes 33

8.1 `chown`: Change file owner and group 33
8.2 `chgrp`: Change group ownership 34
8.3 `chmod`: Change access permissions 34
8.4 `touch`: Change file timestamps 35

9 Disk usage 37

9.1 `df`: Report filesystem disk space usage 37
9.2 `du`: Estimate file space usage 39
9.3 `sync`: Synchronize data on disk with memory 40

Index 41