gnu:guide/fileutils

COLLABORATORS			
	TITLE : gnu:guide/fileutils		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 22, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

gnu:guide/fileutils

1.1 gnu:guide/fileutils.guide

```
GNU file utilities
******************

   This manual minimally documents version GNU fileutils 3.12 of the
GNU file utilities.


  Introduction      Caveats, overview, and authors.
  Common options                Common options.
  File permissions      Access modes.
  Directory listing             ls dir vdir d v
  Basic operations              cp dd install mv rm
  Special file types            ln mkdir rmdir mkfifo mknod
  Changing file attributes      chgrp chmod chown touch
  Disk usage                    df du sync
  Index                         General index.
```

1.2 fileutils.guide/Introduction

```
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 'bug-gnu-utils@prep.ai.mit.edu'.  Remember to
include the version number, machine architecture, input files, and any
other information needed to reproduce the bug.  See Bugs.
```

This manual is based on the Unix man pages in the distribution, which were originally written by David MacKenzie and updated by Jim Meyering. Franc,ois 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.

1.3 fileutils.guide/Common options

Common options
****

Certain options are available in all 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.)

‘--help’
Print a usage message listing all available options, then exit successfully.

‘--version’
Print the version number, then exit successfully.

Backup options -b -S -V, in some programs.

1.4 fileutils.guide/Backup options

Backup options
==========

Some GNU programs (at least ‘cp’, ‘mv’, and ‘ln’) 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 METHODs (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.
```

1.5 fileutils.guide/File permissions

```
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.


  Mode Structure               Structure of file permissions.
  Symbolic Modes               Mnemonic permissions representation.
  Numeric Modes                Permissions as octal numbers.
```

1.6 fileutils.guide/Mode Structure

```
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".

1.7 fileutils.guide/Symbolic Modes

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 Umask and Protection).

The format of symbolic modes is:

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

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

1.8 fileutils.guide/Setting Permissions

```
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 Umask and Protection, 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
```

1.9 fileutils.guide/Copying Permissions

```
Copying Existing Permissions
----------------------------
```

You can base part of a file’s permissions on part of 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.

1.10 fileutils.guide/Changing Special Permissions

```
Changing Special Permissions
----------------------------
```

In addition to changing a file’s read, write, and execute permissions, you can change its special permissions. See Mode Structure, 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 Mode Structure). 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.

1.11 fileutils.guide/Conditional Executability

```
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.

1.12 fileutils.guide/Multiple Changes

```
Making Multiple Changes
-----------------------
```

The format of symbolic modes is actually more complex than described above (see Setting Permissions). 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
```

1.13 fileutils.guide/Umask and Protection

```
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.
```

1.14 fileutils.guide/Numeric Modes

```
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=’.

1.15 fileutils.guide/Directory listing

Directory listing
****

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

1.16 fileutils.guide/ls invocation

‘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 filename. 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.

Also, the ‘-g’ option is accepted but ignored, for compatibility
with Unix. Also see See Common options.

Formatting the filenames

1.17 fileutils.guide/Which files are listed

```
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 filename does not match a
     wildcard at the start of PATTERN.

`-L'
`--dereference'
     List the files linked to by symbolic links instead of listing the
     contents of the links.

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

1.18 fileutils.guide/What information is listed

```
What information is listed
--------------------------
```

```
   These options affect the information that ‘ls’ displays.  By
default, only filenames 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 filename 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 filename.  (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 6 months old or more than 1 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 bug.

     This output format is the default for the GNU ‘v’ and ‘vdir’
     programs.

‘-s’
‘--size’
     Print the size of each file in 1024-byte blocks to the left of the
```

filename. If the environment variable 'POSIXLY_CORRECT' is set, 512-byte blocks are used instead, unless the '-k' option is given (see General output formatting).

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.

1.19 fileutils.guide/Sorting the output

```
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' 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.
```

1.20 fileutils.guide/General output formatting

```
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.

‘-F’
‘--classify’
     Append a character to each filename 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
     ‘date’ invocation.

     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 filename indicating the file type. This
     is like ‘-F’, except that executables are not marked.

‘-x’
‘--format=across’
‘--format=horizontal’
     List the files in columns, sorted horizontally.

‘-T’
‘--tabsize=COLS’
     Assume that each tabstop is COLS columns wide.  The default is 8.
     ‘ls’ uses tabs where possible in the output, for efficiency.

‘-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.
```

1.21 fileutils.guide/Formatting the filenames

```
Formatting the filenames
------------------------

   These options change how filenames themselves are printed.

‘-b’
‘--escape’
     Quote nongraphic characters in filenames using alphabetic and octal
     backslash sequences like those used in C.

‘-N’
‘--literal’
     Do not quote filenames.

‘-q’
‘--hide-control-chars’
     Print question marks instead of nongraphic characters in filenames.
     This is the default.

‘-Q’
‘--quote-name’
```

Enclose filenames in double quotes and quote nongraphic characters
as in C.

1.22 fileutils.guide/dir invocation

'dir': Briefly list directory contents
======================================

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

See 'ls'.

1.23 fileutils.guide/vdir invocation

'vdir': Verbosely list directory contents
===

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

1.24 fileutils.guide/Basic operations

Basic operations
****

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

cp invocation	Copy files.
dd invocation	Convert and copy a file.
install invocation	Copy files and set attributes.
mv invocation	Move (rename) files.
rm invocation	Remove files or directories.

1.25 fileutils.guide/cp invocation

'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.

If the source file contains holes, ‘cp’ copies them and other blocks of zero bytes as holes. Otherwise, files are written just as they are read. (A "hole" is a sequence of zero bytes that does not occupy any physical disk blocks; the ‘read’ system call reads these as zeroes.)

By default, ‘cp’ does not copy directories. It also refuses to copy a file onto itself.

The program accepts the following options. Also see See Common options.

‘-a’
‘--archive’
: Preserve as much as possible of the structure and attributes of the original files in the copy. Equivalent to ‘-dpR’.

‘-b’
‘--backup’
: Make backups of files that are about to be overwritten or removed. See Backup options.

‘-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).

‘-s’
‘--symbolic-link’
Make symbolic links instead of copies of non-directories. All source filenames 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 Backup options.

‘-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 Backup options.

‘-x’
‘--one-file-system’
Skip subdirectories that are on different filesystems from the one that the copy started on.

1.26 fileutils.guide/dd invocation

```
'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 See
Common options.

   The numeric-valued options below (BYTES and BLOCKS) can be followed
by a multiplier: 'b'=512, 'c'=1, 'k'=1024, 'w'=2, 'xM'=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.

1.27 fileutils.guide/install invocation

‘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 See Common options.

'-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 File permissions). 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.

1.28 fileutils.guide/mv invocation

'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 See Common options.

‘-b’
‘--backup’
: Make backups of files that are about to be overwritten or removed. See Backup options.

‘-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 Backup options.

‘-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 Backup options.

1.29 fileutils.guide/rm invocation

```
'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 See
Common options.

'-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.

1.30 fileutils.guide/Special file types

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".

ln invocation	Make links between files.
mkdir invocation	Make directories.
mkfifo invocation	Make FIFOs (named pipes).
mknod invocation	Make block or character special files.
rmdir invocation	Remove empty directories.

1.31 fileutils.guide/ln invocation

‘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 Symbolic Links.

The program accepts the following options. Also see See Common options.

‘-b’
‘--backup’
Make backups of files that are about to be overwritten or removed. See Backup options.

‘-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 used with ‘--force’ and an explicit destination that is a symlink to a directory, remove (or move it with ‘--backup’) that symlink before making any link.

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
     Backup options.

‘-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 Backup options.
```

1.32 fileutils.guide/mkdir invocation

```
‘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 See
Common options.

‘-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 File permissions.

‘-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.
```

1.33 fileutils.guide/mkfifo invocation

```
'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 See
Common options.

'-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
     File permissions.
```

1.34 fileutils.guide/mknod invocation

```
'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 See
Common options.

'-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 File permissions.
```

1.35 fileutils.guide/rmdir invocation

```
'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 See
Common options.

'-p'
'--parents'
     Remove any parent directories that become empty after an argument
     DIRECTORY is removed.

   See rm invocation, for how to remove non-empty directories
(recursively).
```

1.36 fileutils.guide/Changing file attributes

```
Changing file attributes
************************

   Files are not merely contents, a name, and a file type (see
Special file types).  They also have 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 all this a
file's "attributes".
```

```
   These commands change file attributes.


  chown invocation              Change file owners and groups.
  chgrp invocation              Change file groups.
  chmod invocation              Change access permissions.
  touch invocation              Change file timestamps.
```

1.37 fileutils.guide/chown invocation

```
'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 See
Common options.

'-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.

‘-v’
‘--verbose’
     Verbosely describe the action (or non-action) taken for every FILE.

‘-R’
‘--recursive’
     Recursively change ownership of directories and their contents.
```

1.38 fileutils.guide/chgrp invocation

```
‘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 See
Common options.

‘-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.

‘-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.
```

1.39 fileutils.guide/chmod invocation

```
'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 See
Common options.

'-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.
```

1.40 fileutils.guide/touch invocation

```
'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 filename.

   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 See
Common options.

'-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 'date' invocation.

'-f'
     Ignored; for compatibility with BSD versions of 'touch'.

'-m'
'--time=mtime'
'--time=modify'
     Change the modification time only.

'-r REFERENCE-FILE'
'--file=REFERENCE-FILE'
     Use the times of 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.
```

1.41 fileutils.guide/Disk usage

```
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.)
```

```
  df invocation                 Report filesystem disk space usage.
  du invocation                 Estimate file space usage.
  sync invocation     Synchronize memory and disk.
```

1.42 fileutils.guide/df invocation

```
‘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 See
Common options.

‘-a’
‘--all’
     Include in the listing filesystems that have 0 blocks, which are
     omitted by default.  Such filesystems are typically special-purpose
     pseudo-filesystems, such as automounter entries.  Filesystems of
     type "ignore" or "auto", supported by some operating systems, are
     only included in the listing if this option is specified.

‘-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’.

‘--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 the results may be slightly out of date.

‘-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, 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’.

1.43 fileutils.guide/du invocation

```
'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 See
Common options.

'-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.

'-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).

‘-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.
```

1.44 fileutils.guide/sync invocation

```
‘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 Common options).
```

1.45 fileutils.guide/Index

```
Index
*****



  - and Unix rm                          rm invocation
  -all                                   df invocation
  -all                                   Which files are listed
```

-all	du invocation
-almost-all	Which files are listed
-archive	cp invocation
-backup	cp invocation
-backup	Backup options
-backup	mv invocation
-backup	ln invocation
-bytes	du invocation
-changes	chgrp invocation
-changes	chown invocation
-changes	chmod invocation
-classify	General output formatting
-count-links	du invocation
-date	touch invocation
-dereference	du invocation
-dereference	Which files are listed
-dereference-args	du invocation
-directory	install invocation
-directory	Which files are listed
-directory	rm invocation
-directory	ln invocation
-dired	What information is listed
-escape	Formatting the filenames
-exclude-type	df invocation
-file	touch invocation
-force	ln invocation
-force	rm invocation
-force	mv invocation
-force	cp invocation
-format	General output formatting
-format	What information is listed
-format	General output formatting
-format	General output formatting
-format	General output formatting
-full-time	General output formatting
-group	install invocation
-help	Common options
-hide-control-chars	Formatting the filenames
-ignore-backups	Which files are listed
-ignore=PATTERN	Which files are listed
-inode	What information is listed
-inodes	df invocation
-interactive	cp invocation
-interactive	ln invocation
-interactive	rm invocation
-interactive	mv invocation
-kilobytes	df invocation
-kilobytes	du invocation
-kilobytes	General output formatting
-link	cp invocation
-literal	Formatting the filenames
-mode	mkdir invocation
-mode	mknod invocation
-mode	install invocation
-mode	mkfifo invocation
-no-create	touch invocation
-no-dereference	cp invocation

-verbose	cp invocation
-version	Common options
-version-control	cp invocation
-version-control	Backup options
-version-control	ln invocation
-version-control	mv invocation
-width	General output formatting
-1	General output formatting
-A	Which files are listed
-a	Which files are listed
-a	touch invocation
-a	du invocation
-a	cp invocation
-a	df invocation
-B	Which files are listed
-b	Formatting the filenames
-b	ln invocation
-b	Backup options
-b	cp invocation
-b	du invocation
-b	mv invocation
-c	chgrp invocation
-c	du invocation
-c	touch invocation
-c	install invocation
-c	chmod invocation
-c	chown invocation
-c	Sorting the output
-C	General output formatting
-d	ln invocation
-d	touch invocation
-D	du invocation
-d	Which files are listed
-d	cp invocation
-D	What information is listed
-d	install invocation
-d	rm invocation
-f	chgrp invocation
-f	ln invocation
-f	cp invocation
-f	mv invocation
-f	chmod invocation
-f	rm invocation
-f	Sorting the output
-F	ln invocation
-f	chown invocation
-F	General output formatting
-g	install invocation
-G	What information is listed
-g	ls invocation
-i	cp invocation
-I	Which files are listed
-i	mv invocation
-i	rm invocation
-i	ln invocation
-i	What information is listed
-i	df invocation

-k	General output formatting
-k	du invocation
-k	df invocation
-L	Which files are listed
-L	du invocation
-l	cp invocation
-l	du invocation
-l	What information is listed
-m	mkfifo invocation
-m	General output formatting
-m	touch invocation
-m	mkdir invocation
-m	mknod invocation
-m	install invocation
-n	General output formatting
-N	Formatting the filenames
-n	ln invocation
-o	install invocation
-p	mkdir invocation
-p	rmdir invocation
-P	df invocation
-P	cp invocation
-p	cp invocation
-q	Formatting the filenames
-Q	Formatting the filenames
-r	Sorting the output
-r	touch invocation
-r	rm invocation
-R	cp invocation
-R	chgrp invocation
-R	rm invocation
-R	Which files are listed
-R	chown invocation
-R	chmod invocation
-s	cp invocation
-S	ln invocation
-S	Sorting the output
-S	Backup options
-s	What information is listed
-s	du invocation
-S	cp invocation
-S	du invocation
-S	mv invocation
-s	ln invocation
-s	install invocation
-T	General output formatting
-t	df invocation
-T	df invocation
-t	Sorting the output
-u	cp invocation
-U	Sorting the output
-u	mv invocation
-u	Sorting the output
-v	rm invocation
-V	cp invocation
-V	ln invocation
-v	cp invocation