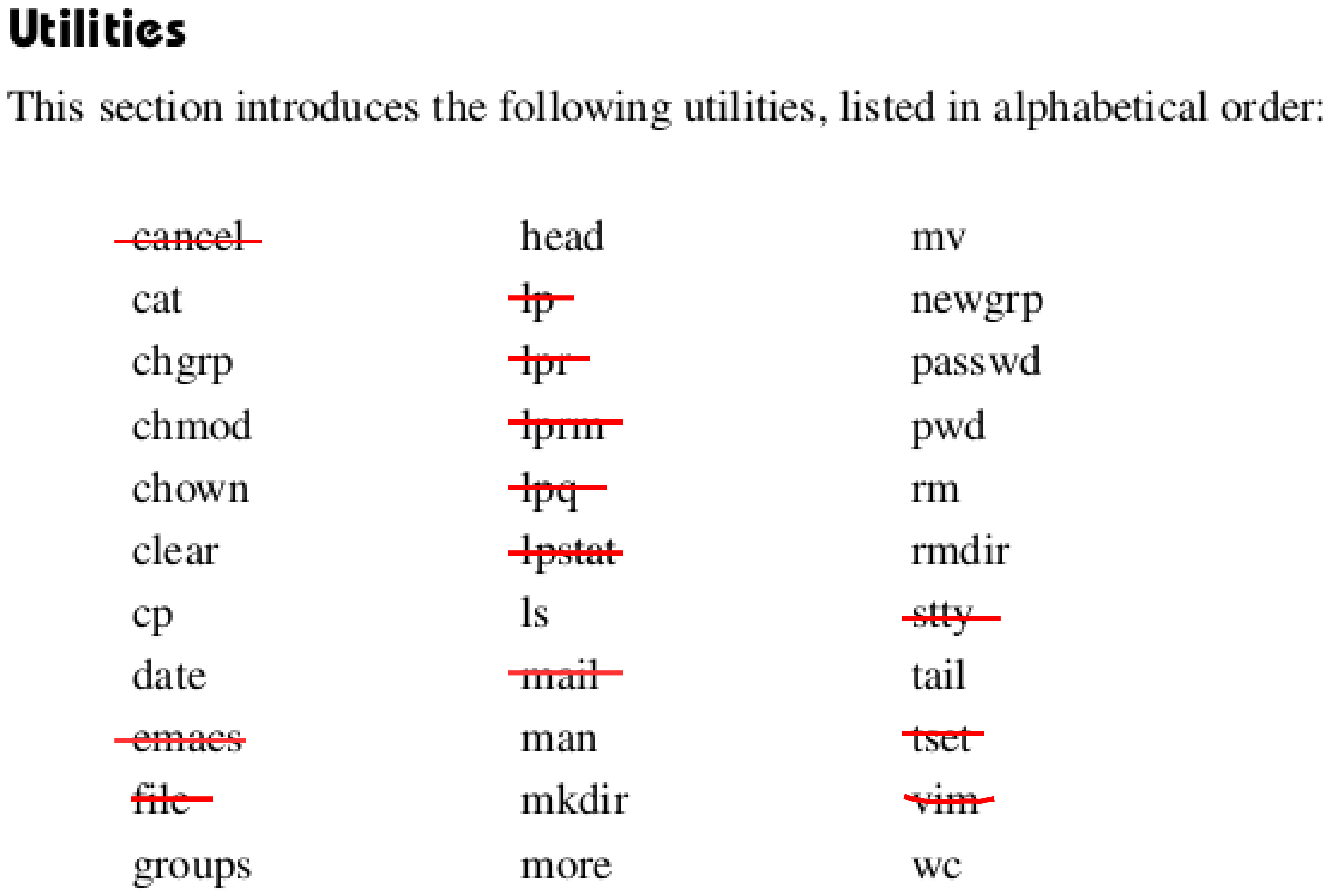
Basic UNIX Commands



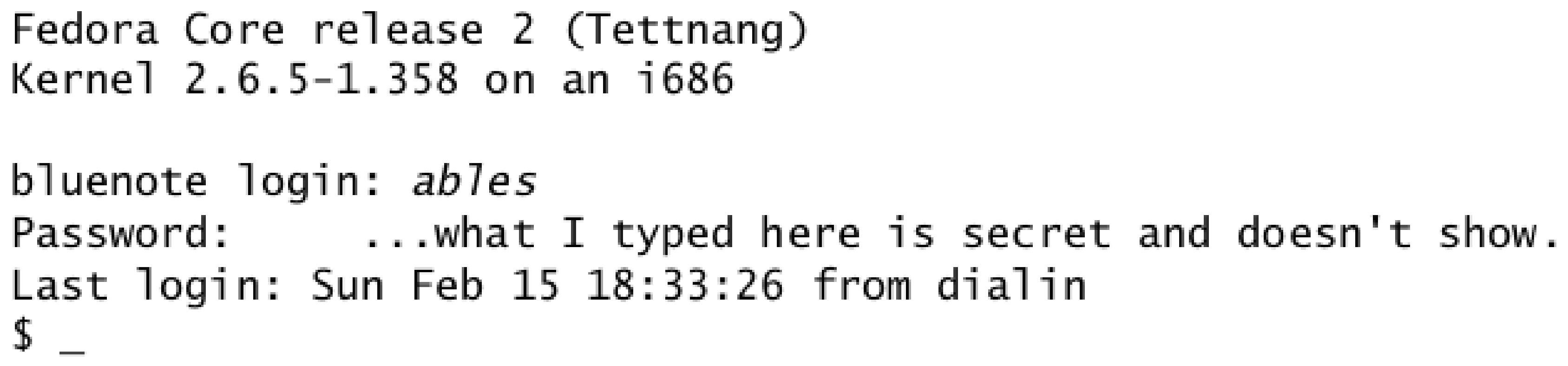
Utilities

This section introduces the following utilities, listed in alphabetical order:

~~cancel~~	head	mv
cat	~~lp~~	newgrp
chgrp	~~lpr~~	passwd
chmod	~~lprm~~	pwd
chown	~~lpq~~	rm
clear	~~lpstat~~	rmdir
cp	ls	~~stty~~
date	~~mail~~	tail
~~emacs~~	man	~~tset~~
~~file~~	mkdir	~~vim~~
groups	more	wc

Logging In

- Your login might look slightly different, depending on how you are connecting. The basic idea is that you must enter a USERID and PASSWORD.



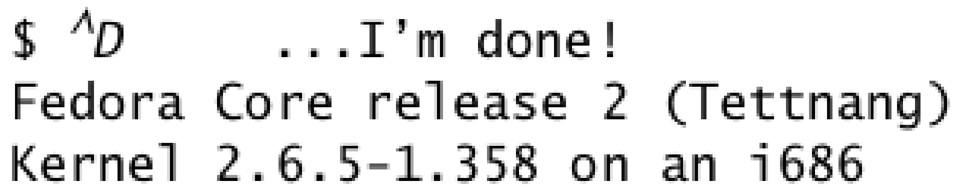
```
Fedora Core release 2 (Tettnang)
Kernel 2.6.5-1.358 on an i686

bluenote login: ables
Password:      ...what I typed here is secret and doesn't show.
Last login: Sun Feb 15 18:33:26 from dialin
$ _
```

$ is the command prompt as used in the book.
Yours might look different, and can be changed.

The shell

- The “shell” is the name of the program that allows you to type in commands and execute them
- There are several different shells – similar, but different in detail.
- Sometimes called a “command interpreter”
- We will use one that is the most popular on Linux and Macs: bash (“Bourne Again Shell”)
- Others are:
  - sh (the original Bourne shell)
  - ksh (Korn shell)
  - csh (C shell)

Logging Out



```
$ ^D      ...I'm done!
Fedora Core release 2 (Tettnang)
Kernel 2.6.5-1.358 on an i686
```

Or type *exit*

Home (login) directory

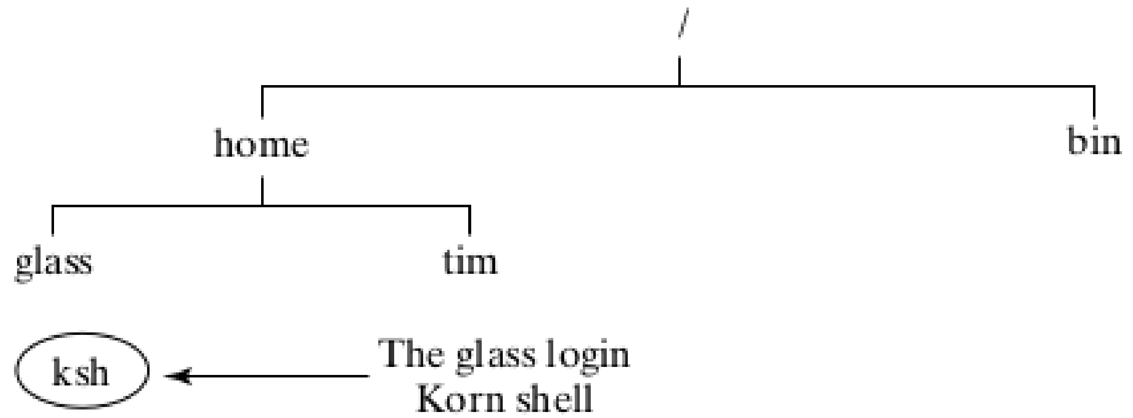


Figure 3–8 The login shell starts at the user's home directory.

Utility: **pwd**

Prints the current working directory.

Figure 3–7 Description of the **pwd** command.

Utility: **passwd**

passwd allows you to change your password. You are prompted for your old password and then twice for the new one (since what you type isn't shown on the screen, you would not know if you made a typo). The new password may be stored in an encrypted form in the password file "/etc/passwd" or in a "shadow" file (for more security) depending on which was selected during the installation of your Linux system.

Figure 3–5 Description of the **passwd** command.

Here's an example, with the passwords shown. Note that you wouldn't normally be able to see the passwords, as Linux turns off the keyboard echo when you enter them.

```
$ passwd
Changing password for user ables.
Changing password for ables
(current) UNIX password: penguin       ... poor choice.
New UNIX password: GWK145W             ... better choice.
Retype new UNIX password: GWK145W
passwd: all authentication tokens updated successfully.
$ _
```

Utility: **date** [*yymmddhhmm* [*.ss*]]

Without any arguments, **date** displays the current date and time. If arguments are provided, **date** sets the date to the supplied setting, where *yy* is the last two digits of the year, the first *mm* is the number of the month, *dd* is the number of the day, *hh* is the number of hours (use the 24-hour clock), and the last *mm* is the number of minutes. The optional *ss* is the number of seconds. Only a super-user may set the date.

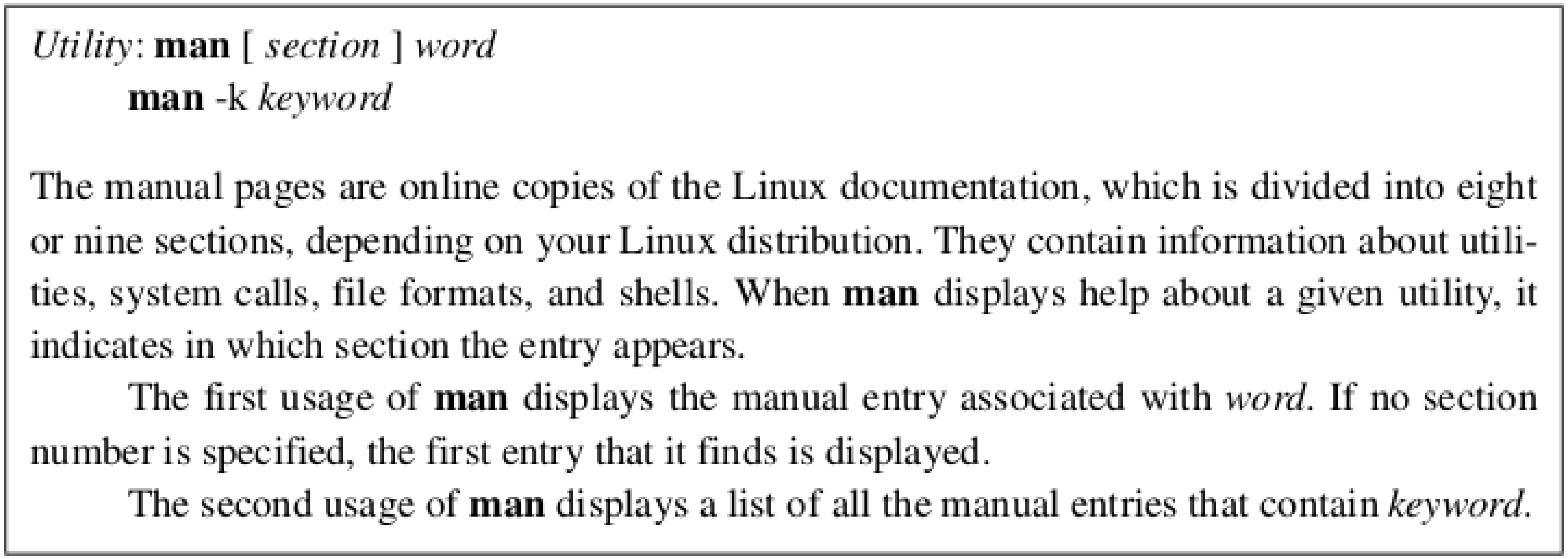
Figure 3–1 Description of the **date** command.

Utility: **clear**

This utility clears your screen.

Figure 3–2 The **clear** command.

Quick Help: man



Utility: **man** [*section*] *word*
man -k *keyword*

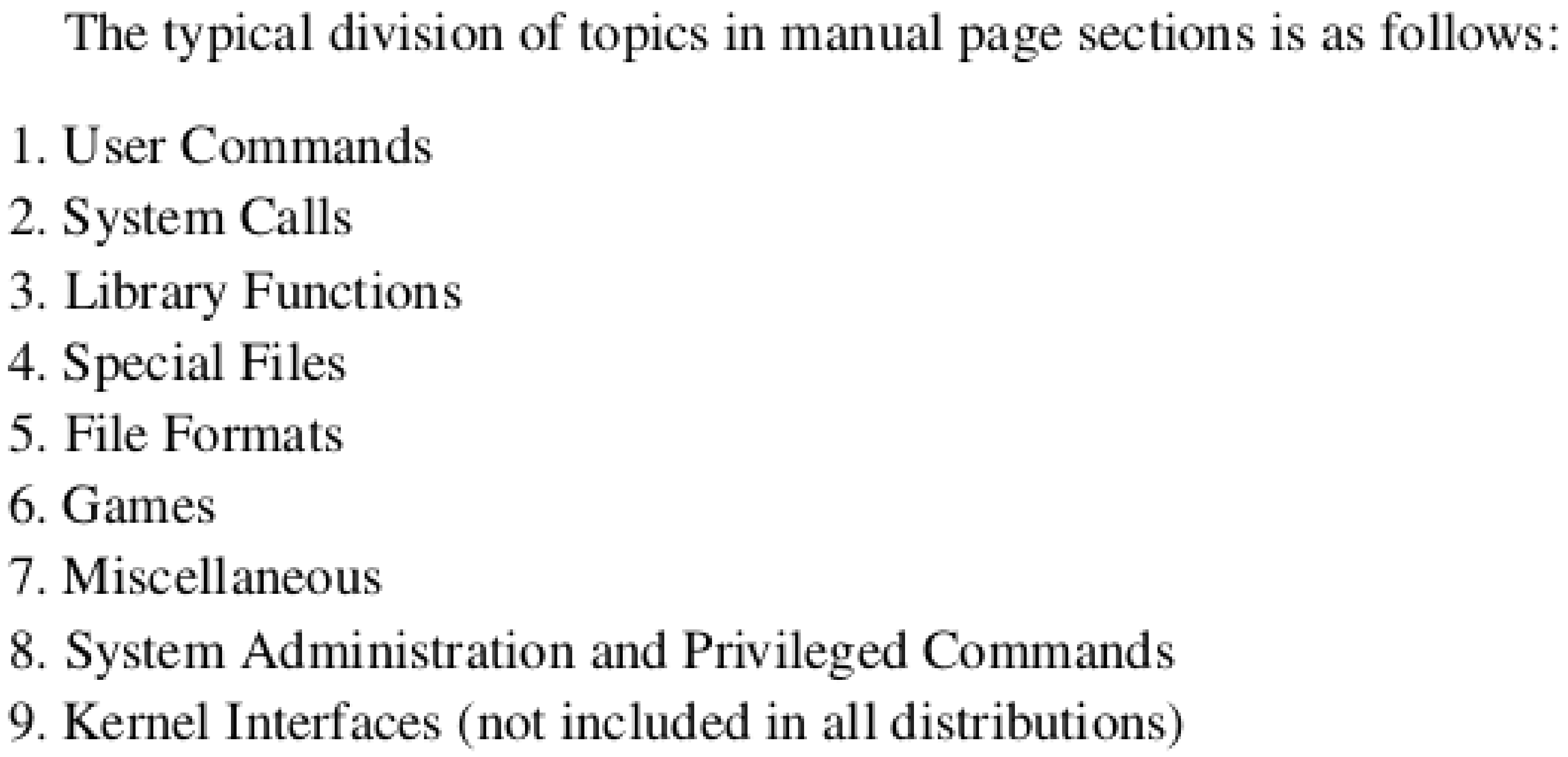
The manual pages are online copies of the Linux documentation, which is divided into eight or nine sections, depending on your Linux distribution. They contain information about utilities, system calls, file formats, and shells. When **man** displays help about a given utility, it indicates in which section the entry appears.

The first usage of **man** displays the manual entry associated with *word*. If no section number is specified, the first entry that it finds is displayed.

The second usage of **man** displays a list of all the manual entries that contain *keyword*.

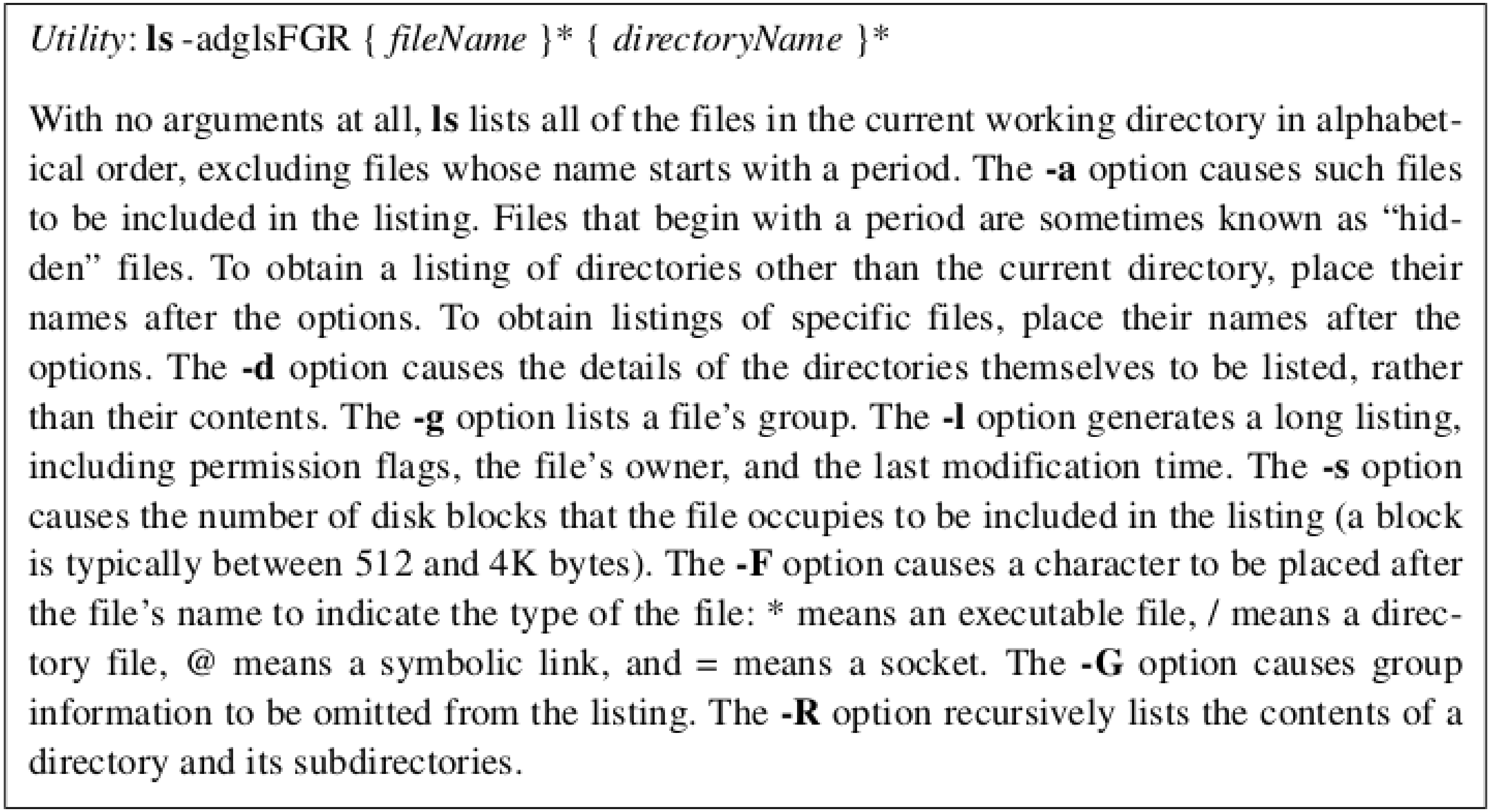
Figure 3–3 The **man** command.

man Sections


The typical division of topics in manual page sections is as follows:

1. User Commands
2. System Calls
3. Library Functions
4. Special Files
5. File Formats
6. Games
7. Miscellaneous
8. System Administration and Privileged Commands
9. Kernel Interfaces (not included in all distributions)

Listing files



Utility: **ls** -adglsFGR { *fileName* }* { *directoryName* }*

With no arguments at all, **ls** lists all of the files in the current working directory in alphabetical order, excluding files whose name starts with a period. The **-a** option causes such files to be included in the listing. Files that begin with a period are sometimes known as "hidden" files. To obtain a listing of directories other than the current directory, place their names after the options. To obtain listings of specific files, place their names after the options. The **-d** option causes the details of the directories themselves to be listed, rather than their contents. The **-g** option lists a file's group. The **-l** option generates a long listing, including permission flags, the file's owner, and the last modification time. The **-s** option causes the number of disk blocks that the file occupies to be included in the listing (a block is typically between 512 and 4K bytes). The **-F** option causes a character to be placed after the file's name to indicate the type of the file: * means an executable file, / means a directory file, @ means a symbolic link, and = means a socket. The **-G** option causes group information to be omitted from the listing. The **-R** option recursively lists the contents of a directory and its subdirectories.

Figure 3–14 Description of the **ls** command.

```
$ ls              ...list all files in current directory.
heart
$ ls -lG heart     ...long listing of 'heart'.
-rw-r--r--  1    glass     106    Jan 30 19:46    heart
$ _
```

Field #	Field value	Meaning
1	-rw-r--r--	The type and permission mode of the file, which indicates who can read, write, and execute the file.
2	1	The hard link count (discussed much later in this book).
3	glass	The username of the owner of the file.
4	106	The size of the file (in bytes).
5	Jan 30 19:46	The time that the file was last modified.
6	heart	The name of the file.

Figure 3–15 Description of output from the **ls** command.

```
$ ls -alFs        ...extra-long listing of current dir.
total 3            ...total number of blocks of storage.
1 drwxr-xr-x  3   glass    cs     512   Jan 30 22:52 ./
1 drwxr-xr-x 12   root     cs    1024   Jan 30 19:45 ../
1 -rw-r--r--  1   glass    cs     106   Jan 30 19:46 heart
$ _
```

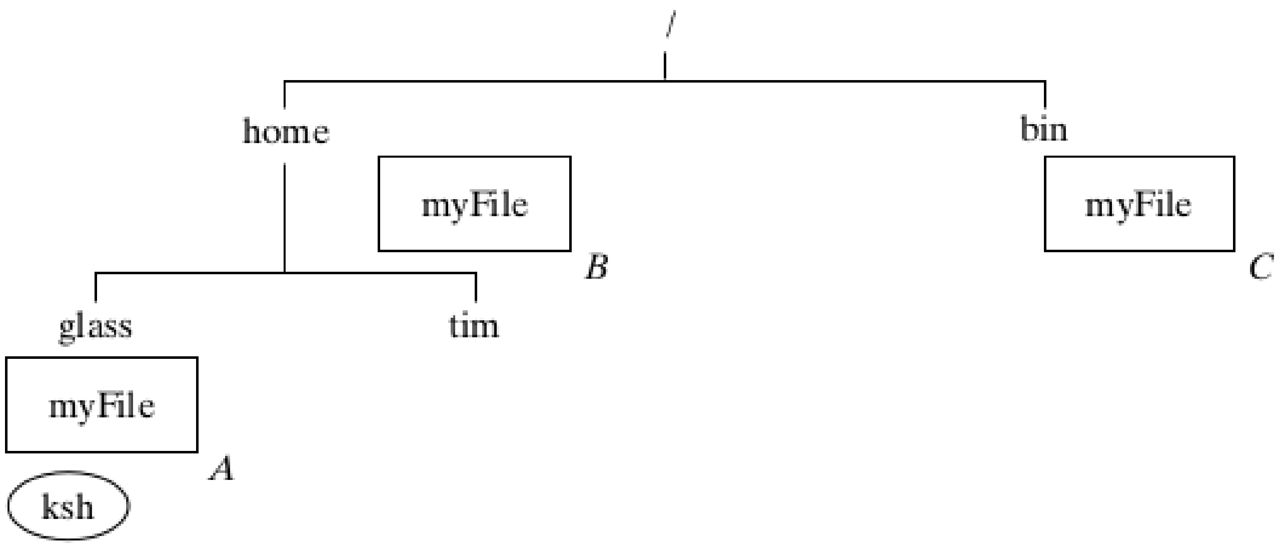


Figure 3–9 Different files may have the same name.

File	Absolute PathName
A	/home/glass/myFile
B	/home/myFile
C	/bin/myFile

Figure 3–10 Absolute pathnames.

Field	Meaning
.	current directory
..	parent directory

Figure 3–11 Current and parent directories.

File	Relative Pathname
A	myFile
B	../myFile
C	../../bin/myFile

Figure 3–12 Relative pathnames.

```
$ ls -lsF heart.final
1 -rw-r--r--   1   glass  cs   213   Jan 31 00:12   heart.final
$ _
```

Each field is the value of a file attribute, described by the Figure 3–33.

Field #	Field value	Meaning
1	1	The number of blocks of physical storage occupied by the file.
2	-rw-r--r--	The type and permission mode of the file—which indicates who can read, write, and execute the file.
3	1	The hard link count (discussed in Chapter 4, "GNU Utilities for Power Users").
4	glass	The username of the owner of the file.
5	cs	The group name of the file.
6	213	The size of the file, in bytes.
7	Jan 31 00:12	The date and time that the file was last modified.
8	heart.final	The name of the file.

Figure 3–33 File attributes.

User (owner)	Group	Others
rw-	r--	r--
Read permission	**Write permission**	**Execute permission**
r	w	x

Figure 3–36 File permissions.

	Regular file	Directory file	Special file
read	The process may access the contents.	The process may read the directory (i.e., list the names of the files that it contains).	The process may read from the file using the read () system call.
write	The process may change the contents.	The process may add or remove files to/from the directory.	The process may write to the file using the write () system call.
execute	The process may execute the file (which only makes sense if it's a program).	The process may access files in the directory or any of its subdirectories.	No meaning.

Figure 3–37 Permission meanings for file types.

Character	File type
-	regular file
d	directory file
b	buffered (block-oriented) special file (such as a disk drive)
c	unbuffered (character-oriented) special file (such as a terminal)
l	symbolic link
p	pipe
s	socket

Figure 3–34 File types.

Typing files

Utility: **cat** -n { *fileName* }*

The **cat** utility takes its input from standard input or from a list of files and displays them to standard output. The **-n** option adds line numbers to the output. **cat** is short for "concatenate," which means "to connect in a series of links."

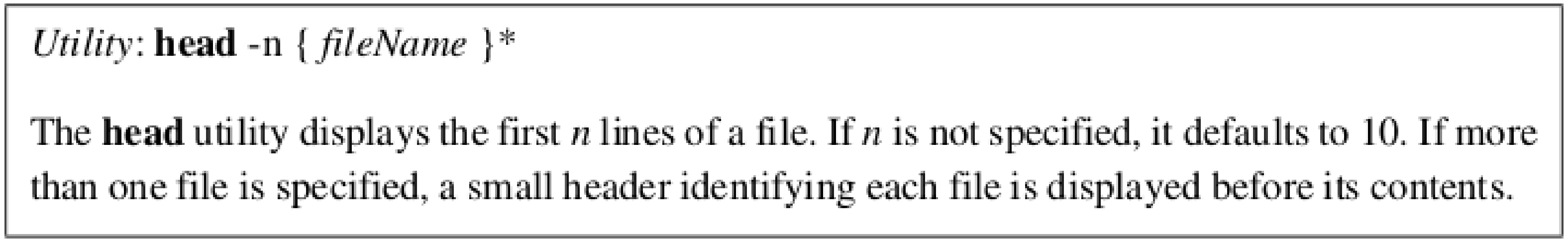
Figure 3–13 Description of the **cat** command.

```
$ cat heart    ...list the contents of the 'heart' file.
I hear her breathing,
I'm surrounded by the sound.
Floating in this secret place,
I never shall be found.
$ _
```

Utility: **more** -f [+*lineNumber*] { *fileName* }*

The **more** utility allows you to scroll through a list of files, one page at a time. By default, each file is displayed starting at line 1, although the + option may be used to specify the starting line number. The **-f** option tells **more** not to fold long lines. After each page is displayed, **more** displays the message "-- More --" to indicate that it's waiting for a command. To list the next page, press the space bar. To list the next line, press the *Enter* key. To quit from **more**, press the *q* key. To obtain help on the multitude of other commands, press the *h* key.

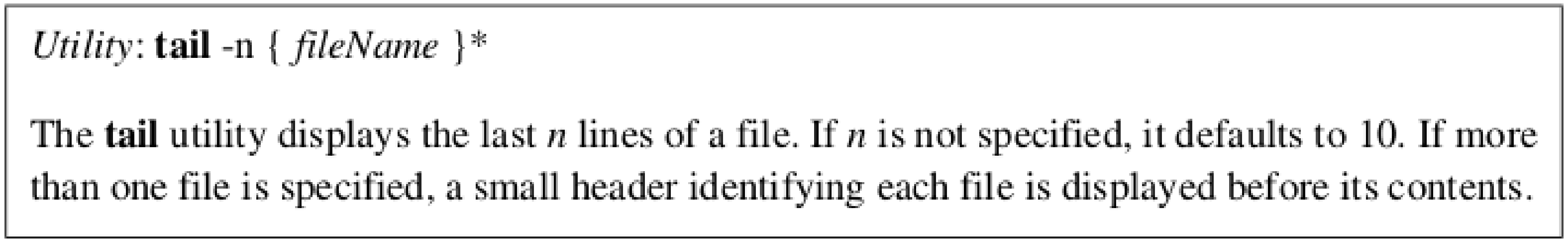
Figure 3–16 Description of the **more** command.



Utility: **head** -n { *fileName* }*

The **head** utility displays the first *n* lines of a file. If *n* is not specified, it defaults to 10. If more than one file is specified, a small header identifying each file is displayed before its contents.

Figure 3–17 Description of the **head** command.

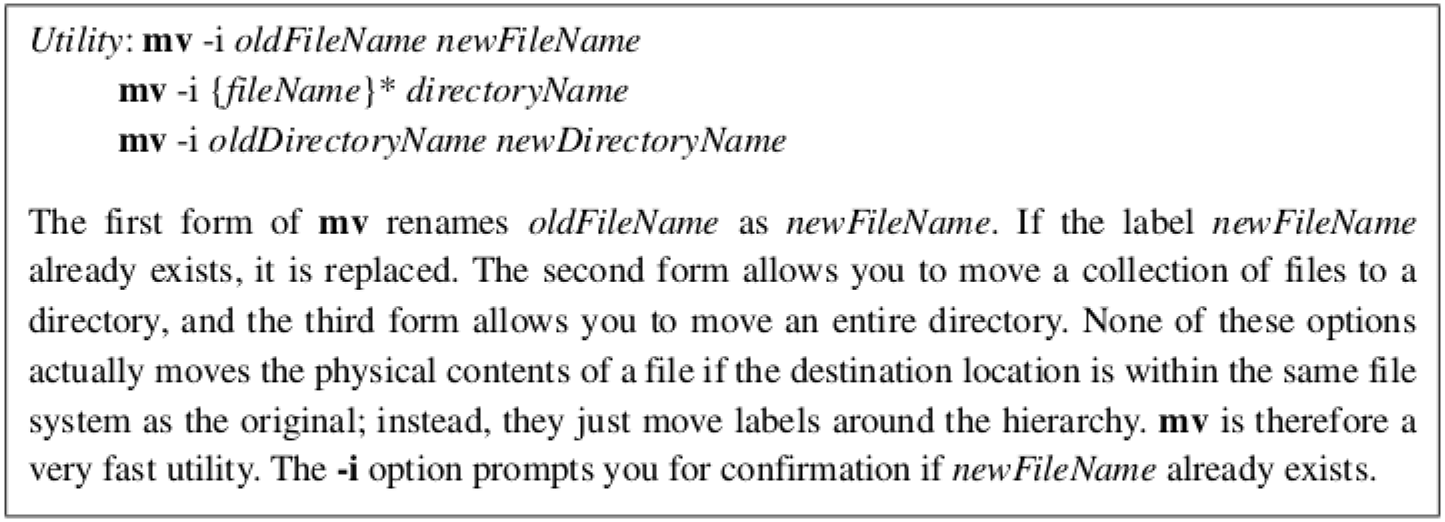


Utility: **tail** -n { *fileName* }*

The **tail** utility displays the last *n* lines of a file. If *n* is not specified, it defaults to 10. If more than one file is specified, a small header identifying each file is displayed before its contents.

Figure 3–18 Description of the **tail** command.

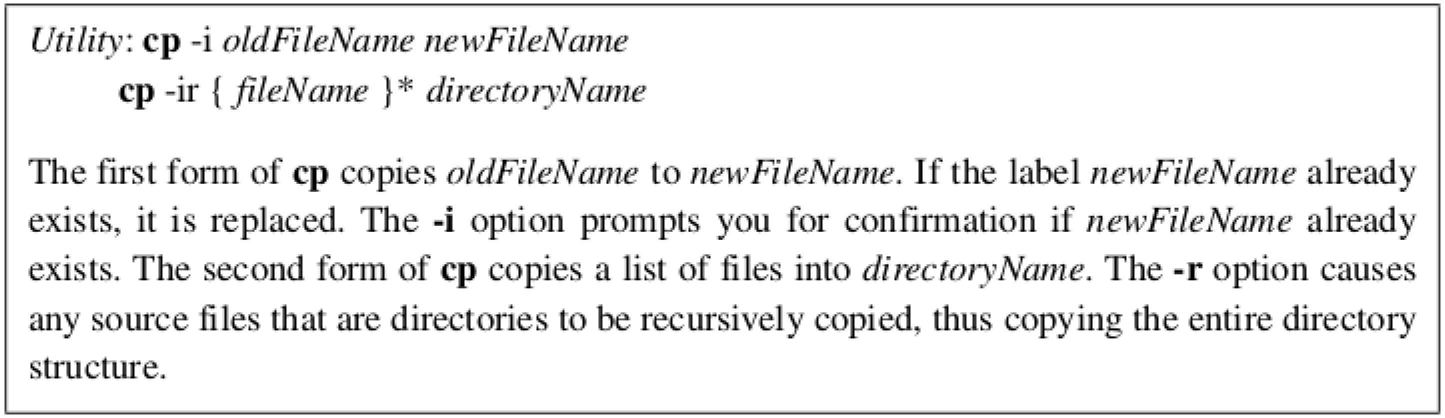
File Operations



Utility: **mv** -i *oldFileName newFileName*
mv -i {*fileName*}* *directoryName*
mv -i *oldDirectoryName newDirectoryName*

The first form of **mv** renames *oldFileName* as *newFileName*. If the label *newFileName* already exists, it is replaced. The second form allows you to move a collection of files to a directory, and the third form allows you to move an entire directory. None of these options actually moves the physical contents of a file if the destination location is within the same file system as the original; instead, they just move labels around the hierarchy. **mv** is therefore a very fast utility. The **-i** option prompts you for confirmation if *newFileName* already exists.

Figure 3–19 Description of the **mv** command.



Utility: **cp** -i *oldFileName newFileName*
cp -ir { *fileName* }* *directoryName*

The first form of **cp** copies *oldFileName* to *newFileName*. If the label *newFileName* already exists, it is replaced. The **-i** option prompts you for confirmation if *newFileName* already exists. The second form of **cp** copies a list of files into *directoryName*. The **-r** option causes any source files that are directories to be recursively copied, thus copying the entire directory structure.

Figure 3–23 Description of the **cp** command.

Utility: **rm** -fir {*fileName*} *

The **rm** utility removes a file's label from the directory hierarchy. If the filename doesn't exist, an error message is displayed. The **-i** option prompts the user for confirmation before deleting a filename; press **y** to confirm, and **n** otherwise. If *fileName* is a directory, the **-r** option causes all of its contents, including subdirectories, to be recursively deleted. The **-f** option inhibits all error messages and prompts.

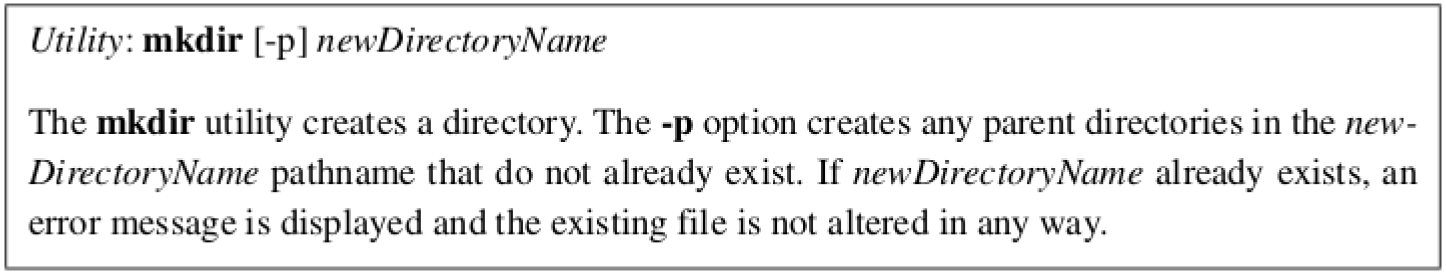
Figure 3–25 Description of the **rm** command.

Utility: **wc** -lwc { *fileName* }*

The **wc** utility counts the lines, words, and/or characters in a list of files. If no files are specified, standard input is used instead. The **-l** option requests a line count, the **-w** option requests a word count, and the **-c** option requests a character count. If no options are specified, then all three counts are displayed. A word is defined by a sequence of characters surrounded by tabs, spaces, or newlines.

Figure 3–32 Description of the **wc** command.

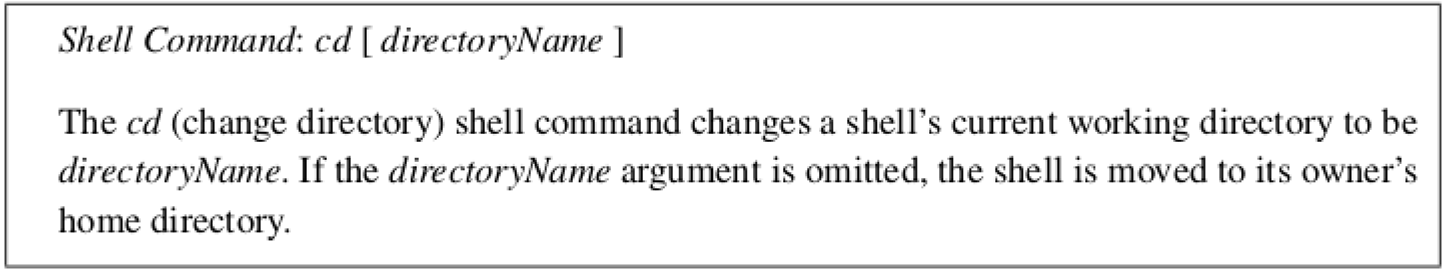
Directory Operations



Utility: **mkdir** [-p] *newDirectoryName*

The **mkdir** utility creates a directory. The **-p** option creates any parent directories in the *new-DirectoryName* pathname that do not already exist. If *newDirectoryName* already exists, an error message is displayed and the existing file is not altered in any way.

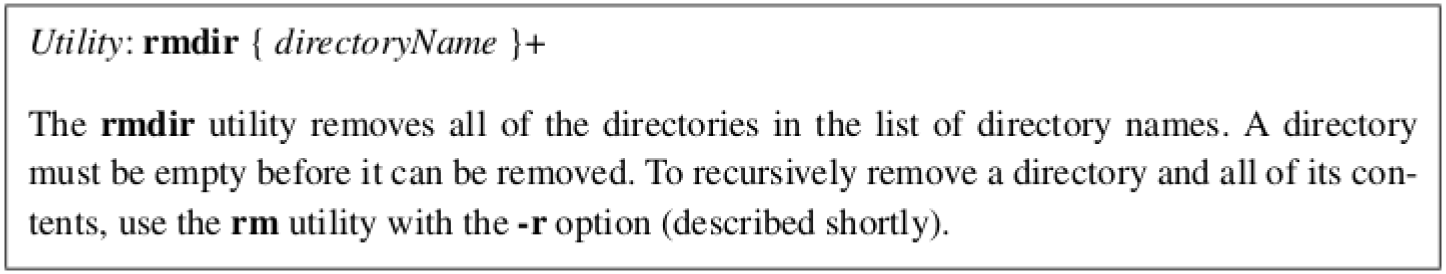
Figure 3–20 Description of the **mkdir** command.



Shell Command: *cd* [*directoryName*]

The *cd* (change directory) shell command changes a shell's current working directory to be *directoryName*. If the *directoryName* argument is omitted, the shell is moved to its owner's home directory.

Figure 3–21 Description of the *cd* shell command.



Utility: **rmdir** { *directoryName* }+

The **rmdir** utility removes all of the directories in the list of directory names. A directory must be empty before it can be removed. To recursively remove a directory and all of its contents, use the **rm** utility with the **-r** option (described shortly).

Figure 3–24 Description of the **rmdir** command.

File Groups

Utility: **chgrp** -R *groupname* { *fileName* }*

The **chgrp** utility allows a user to change the group of files that he/she owns. A super-user can change the group of any file. All of the files that follow the *groupname* argument are affected. The **-R** option recursively changes the group of the files in a directory.

Figure 3–39 Description of the **chgrp** command.

Utility: **groups** [*userId*]

When invoked with no arguments, the **groups** utility displays a list of all the groups that you are a member of. If the name of a user is specified, a list of that user's groups are displayed.

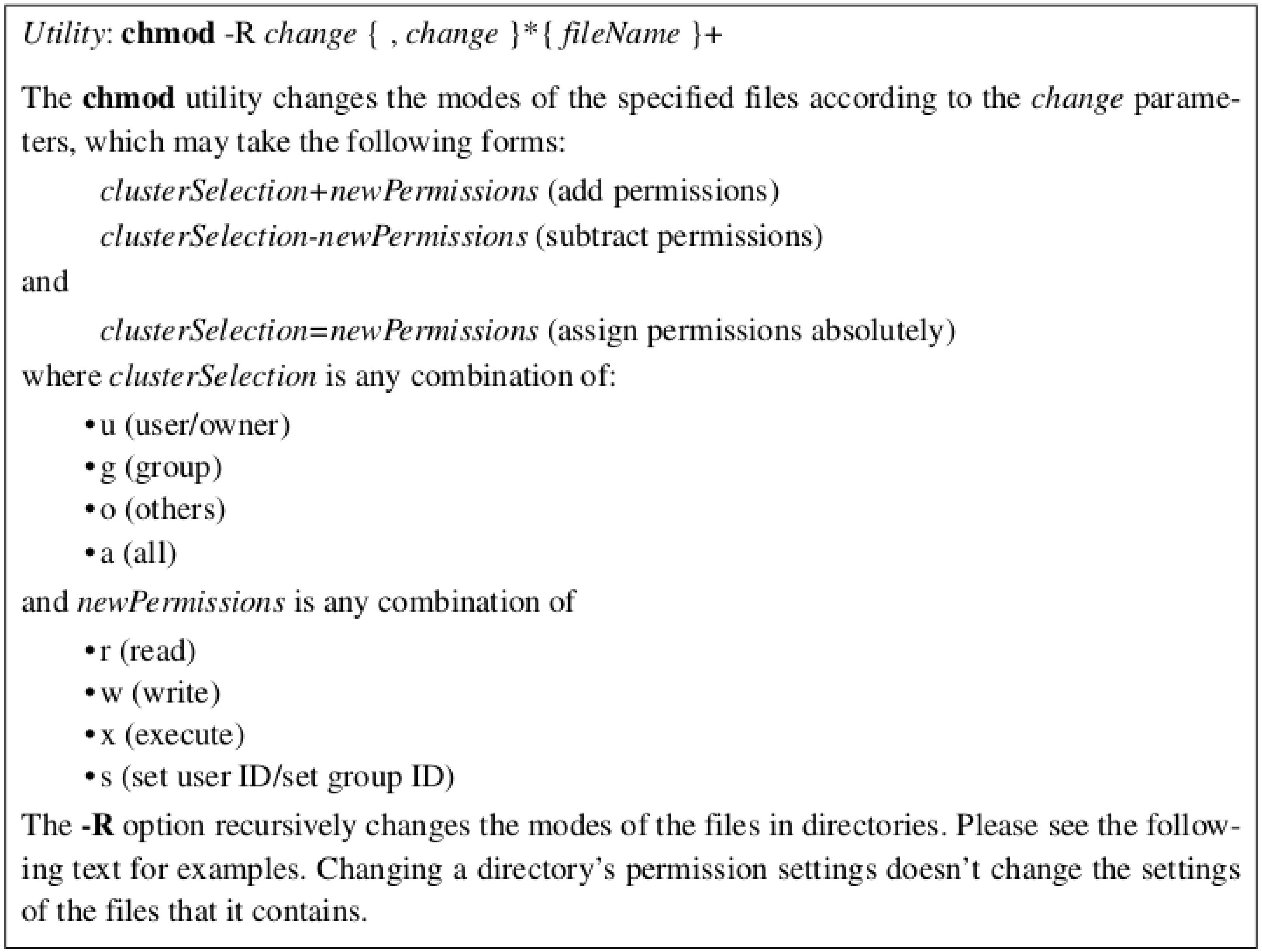
Figure 3–38 Description of the **groups** command.

Utility: **newgrp** { - | *groupname* }

The **newgrp** utility, when invoked with a group name as an argument, creates a new shell with an effective group ID corresponding to the group name. The old shell sleeps until you exit the newly created shell. You must be a member of the group that you specify. If you use a dash (-) instead of a group name as the argument, a shell is created with the same settings as the shell that was created when you logged into the system.

Figure 3–44 Description of the **newgrp** command.

Changing File Attributes



Utility: **chmod** -R *change* { , *change* }*{ *fileName* }+

The **chmod** utility changes the modes of the specified files according to the *change* parameters, which may take the following forms:

clusterSelection+newPermissions (add permissions)

clusterSelection-newPermissions (subtract permissions)

and

clusterSelection=newPermissions (assign permissions absolutely)

where *clusterSelection* is any combination of:

- u (user/owner)
- g (group)
- o (others)
- a (all)

and *newPermissions* is any combination of

- r (read)
- w (write)
- x (execute)
- s (set user ID/set group ID)

The **-R** option recursively changes the modes of the files in directories. Please see the following text for examples. Changing a directory's permission settings doesn't change the settings of the files that it contains.

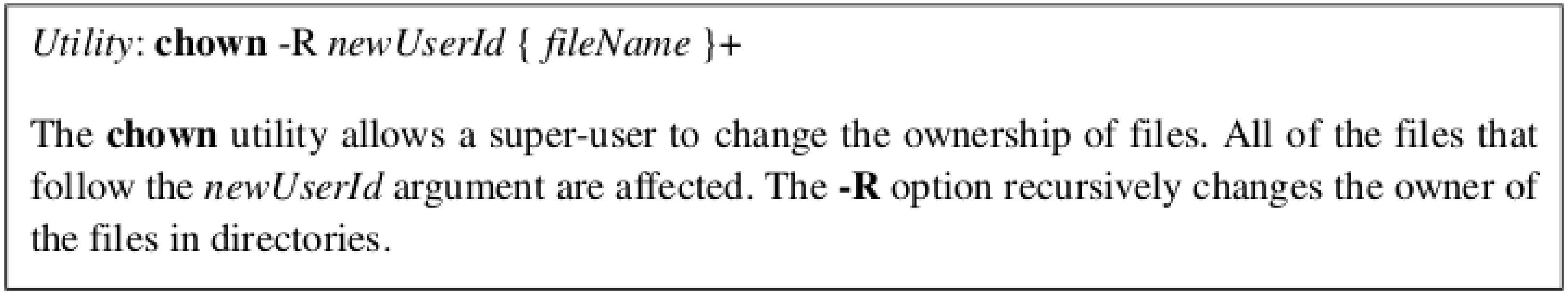
Figure 3–40 Description of the **chmod** command.

	User	Group	Others
setting	rwx	r-x	---
binary	111	101	000
octal	7	5	0

Figure 3–42 Permission of 750 for the **chmod** command.

Requirement	Change parameters
Add group write permission.	g+w
Remove user read and write permission.	u-rw
Add execute permission for user, group, and others.	a+x
Give the group just read permission.	g=r
Add write permission for user, and remove read from group.	u+w, g-r

Figure 3–41 File permission specifications for the **chmod** command.



Utility: **chown** -R *newUserId* { *fileName* }+

The **chown** utility allows a super-user to change the ownership of files. All of the files that follow the *newUserId* argument are affected. The **-R** option recursively changes the owner of the files in directories.

Figure 3–43 Description of the **chmod** command.

Editors

- An editor is used to create an (ASCII text) file
- Generally, the editor you use is independent of the application (although sometimes they can help by syntax coloring).
- UNIX supports several editors:
  - vi (or vim)
  - emacs
  - nano (like wordpad on windows)
- “The best editor to use is the one you know”
- However, knowing a good programming editor is worth knowing, will save you time in the long run.