Linux provides two user interfaces: the graphical user interface (GUI) hosted by X and an older, command-line interface (CLI) called the shell.

Those familiar with the MS-DOS command-line interface will recognize the shell, which you use by typing text commands to which the system responds by displaying text replies. But the comparison with the MS-DOS command line doesn’t do justice to the Linux shell, which is vastly more powerful. And, older doesn’t necessarily imply inferior.

GUIs are stylish primarily because they’re easy to learn and use. But they’re not always the most efficient way of operating a computer. A skilled user of the shell can often outpace a competitor using a GUI. Moreover, a GUI enables its user to perform only the functions provided by the GUI’s programmers. In contrast, the shell is expandable. The shell enables users to define entirely new operations based on sequences of existing operations.

The real power of Linux lies in the shell. So, if you aspire to master Linux, you must conquer the shell. Even if your ambition falls short of gurudom, you’ll find knowledge of the shell helpful. Many procedures from sources other than this book assume that you know how to use the shell. And, if X fails, you can’t easily repair it without knowing how to use the shell.

Linux supports a variety of shells, but the most popular is the bash shell, described in this chapter. The chapter explains how to issue shell commands, and how to use shell commands to manipulate files and directories, work with removable media, and launch programs. The chapter also explains nano, a simple text editor that operates in text mode.

**Issuing Shell Commands**

The most common way to access the shell is via a terminal window, as explained in Chapter 5 and Chapter 6. However, a terminal window isn’t the only way to access
the shell. The “Using virtual consoles” section of Chapter 4 explains how to access
the shell with a virtual console.

As your first Linux command, launch a terminal window, type w, and press Enter.
Your contents of the terminal window should look something like this:

```
[bill@home bill]$ w
11:12am  up 6 min,  1 user,  load average: 0.00, 0.08, 0.05
USER     TTY      FROM      LOGIN@   IDLE   JCPU   PCPU  WHAT
bill     tty1              11:11am  0.00s  0.20s  0.11s  -bash
```

The `w` command tells Linux to display the system status and a list of all system users.
In the example, the output of the command tells you that it's now 11:12 a.m., that
the system has been up for six minutes, and that only one user—bill—is currently
logged in. Notice that the command output is very terse, packing much information
into a few lines. Such output is typical of Linux commands. At first, you may find
Linux output cryptic and difficult to read, but over time you'll grow to appreciate the
efficiency with which Linux communicates information.

Linux command output is not terse owing to an oversight or laziness
on the part of the creators of Linux. Instead, Linux command output
is designed so that it can be processed by programs as well as by
humans. The structure of the output simplifies the task of program-
mers who write programs to process command output.

Linux provides many commands besides the `w` command—so many that you may
despair of learning and recalling them. Actually, the number of commands you'll use
regularly is fairly small. Soon, they will become second nature to you.

Try a second command, the `date` command:

```
[bill@home bill]$ date
Fri Oct 5 11:15:20 PST 2004
```

The `date` command displays the current date and time.

If you find working with MS-DOS distasteful or intimidating, you may not immedi-
ately enjoy working with the Linux command line. However, give yourself some time
to adjust. The Linux command line has several features that make it easier to use,
and more powerful, than MS-DOS.

**Correcting Commands**

Sometimes you may type a command incorrectly, causing Linux to display an error
message. For example, suppose you typed `dat` instead of `date`:

```
[bill@home bill]$ dat
bash: dat: command not found
```
In such a case, carefully check the spelling of the command and try again. If you notice an error before pressing **Enter**, you can use the **Backspace** or **Left** arrow key to return to the point of the error and then type the correct characters. The **Backspace** key erases characters whereas the **Left** arrow key does not. You can also use the **Delete** key to delete unwanted characters.

Just as a web browser keeps track of recently visited sites, the *bash* shell keeps track of recently issued commands in what’s known as the **history list**. You can scroll back through *bash*’s history by using the **Up** arrow key, or back down using the **Down** arrow key, just as you would with the Back and Forward buttons on a web browser. To reissue a command, scroll to it and press **Enter**. If you like, you can modify the command before reissuing it. When typing shell commands, you have access to a minieditor that resembles the DOSKEY editor of MS-DOS. This minieditor lets you revise command lines by typing key commands. Table 7-1 summarizes some useful key commands interpreted by the shell. The key commands let you access a list of the 500 most recently executed commands. *bash*’s history is saved in the history file stored in the user’s home directory, ~/.bash_history.

**Table 7-1. Useful editing keystrokes**

<table>
<thead>
<tr>
<th>Keystroke(s)</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up arrow</td>
<td>Move back one command in the history list.</td>
</tr>
<tr>
<td>Down arrow</td>
<td>Move forward one command in the history list.</td>
</tr>
<tr>
<td>Left arrow</td>
<td>Move back one character.</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Move forward one character.</td>
</tr>
<tr>
<td>Backspace</td>
<td>Delete previous character.</td>
</tr>
<tr>
<td>Tab</td>
<td>Attempt to complete the current word, interpreting it as a filename or command determined by context.</td>
</tr>
<tr>
<td>Alt-B</td>
<td>Move back one word.</td>
</tr>
<tr>
<td>Alt-D</td>
<td>Delete current word.</td>
</tr>
<tr>
<td>Alt-F</td>
<td>Move forward one word.</td>
</tr>
<tr>
<td>Ctrl-A</td>
<td>Move to beginning of line.</td>
</tr>
<tr>
<td>Ctrl-D</td>
<td>Delete current character.</td>
</tr>
<tr>
<td>Ctrl-E</td>
<td>Move to end of line.</td>
</tr>
<tr>
<td>Ctrl-K</td>
<td>Delete to end of line.</td>
</tr>
<tr>
<td>Ctrl-L</td>
<td>Clear the screen, placing the current line at the top of the screen.</td>
</tr>
<tr>
<td>Ctrl-U</td>
<td>Delete from beginning of line.</td>
</tr>
<tr>
<td>Ctrl-Y</td>
<td>Retrieve last item deleted.</td>
</tr>
<tr>
<td>Esc .</td>
<td>Insert last word of previous command (Esc is pressed before the dot, rather than at the same time).</td>
</tr>
<tr>
<td>Esc ? or Tab</td>
<td>List the possible completions (note that Esc is pressed before the question mark, not at the same time).</td>
</tr>
</tbody>
</table>

One of the most useful editing keystrokes, **Tab**, can also be used when typing a command. If you type the first part of a filename and press **Tab**, the shell will attempt to
locate files with names matching the characters you’ve typed. If something exists, the shell fills out the partially typed name with the proper characters. You can then press Enter to execute the command or continue typing other options and arguments. This feature, called either filename or command completion, makes the shell much easier to use.

In addition to keystrokes for editing the command line, the shell interprets several keystrokes that control the operation of the currently executing program. Table 7-2 summarizes these keystrokes. For example, typing Ctrl-C generally cancels execution of a program. This keystroke command is handy, for example, when a program is taking too long to execute and you’d prefer to try something else.

Table 7-2. Useful Control keystrokes

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-C</td>
<td>Sends an interrupt signal to the currently executing command, which generally responds by terminating itself.</td>
</tr>
<tr>
<td>Ctrl-D</td>
<td>Sends an end-of-file to the currently executing command; use this keystroke to terminate console input.</td>
</tr>
<tr>
<td>Ctrl-Z</td>
<td>Suspends the currently executing program. The fg command resumes execution of the suspended program.</td>
</tr>
</tbody>
</table>

Several other special characters control the operation of the shell, as shown in Table 7-3. The # and ; characters are most often used in shell scripts, which you’ll learn about in more detail later in this chapter. The & character causes the shell prompt to return immediately instead of waiting for a command to finish; the command runs in the background and you can continue to enter more commands. The pipe redirector (|) is explained in the section titled “Displaying Directory Contents.”

Table 7-3. Other special shell characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Marks the line as a comment, which the shell ignores.</td>
</tr>
<tr>
<td>;</td>
<td>Separates commands, letting you enter several commands on a single line.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Placed at the end of a command, causes the command to execute as a background process, so that a new shell prompt appears immediately after the command is entered.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Stores output from the command in the file whose name follows.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Takes input from the file whose name follows.</td>
</tr>
<tr>
<td>\</td>
<td>At end of line, continues command on the following line.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Working with the Linux Command Prompt

Linux commands share a simple, common structure. This section describes their common structure and explains how you can obtain helpful information about the commands available to you.
Commands and Arguments

The general form of a shell command line is this:

```
command [options] [arguments]
```

The `command` determines what operation the shell will perform and the `options` and `arguments` customize, or fine-tune, the operation. The `options` and `arguments` may or may not appear, as indicated by the square brackets. Sometimes the `command` specifies a program file that will be launched and run; such a command is called an `external command`. Linux generally stores these files in `/bin`, `/usr/bin`, or `/usr/local/bin`. System administration commands are generally stored in `/sbin` or `/usr/sbin`, which are included by default in the path of the root user. When a command specifies a program file, the shell passes any specified arguments to the program, which scans and interprets them, adjusting its operation accordingly.

Some commands are not external program files; instead they are built-in commands interpreted by the shell itself. One important way in which shells differ is in the built-in commands that they support. Later in this section, you’ll learn about some of bash’s built-in commands.

The name of a Linux command almost always consists of lowercase letters and digits. Most commands let you specify options or arguments. However, in any given case, you may not need to do so. For example, typing the `w` command without options and arguments causes Linux to display a list of current users.

```
Remember, Linux commands are case sensitive; be sure to type each character of a command in the proper case.
```

Options modify the way that a command works. Many options consist of a single letter, prefixed by a dash. Often, you can specify more than one option; when you do so, you separate each option with one or more spaces. For example, the `-h` option of the `w` command causes the output of the command to omit the header lines that give the time and the names of the fields:

```
[bill@home bill]$ w -h
```

Arguments specify filenames or other targets that direct the action of the command. For example, the `w` command lets you specify a username as an argument, which causes the command to list logins that pertain only to the specified user:

```
[bill@home bill]$ w bill
```

Some commands let you specify a series of arguments; you must separate each argument with a space between them. For example, the following command prints a list of logins by the root user, without header lines:

```
[bill@home bill]$ w -h bill
```
When a command includes several arguments, a command may not fit on a single line. However, you can continue typing when you reach the end of a line, because the shell will automatically wrap your input to the next line. If you find line wrapping disconcerting, you can type a backslash (\) at the end of a line, press Enter, and continue typing on the next line. The backslash is the shell’s line continuation character; the shell sees lines joined by a backslash as though they were a single line. Don’t type anything after the backslash or the continuation feature won’t work correctly.

**Getting Help**

Because Linux provides so many commands and because Linux commands provide so many possible options, you can’t expect to recall all of them. To help you, Linux provides the **man** and **apropos** commands, which let you access a help database that describes commands and their options.

**Using man**

Each Linux command is described by a special file called a *manual page*. The manual pages (or *manpages*) are stored in a group of subdirectories comprising a help database. To access this database, you use the **man** command, which resembles the MS-DOS **help** command. For example, to get help on using the **w** command, type:

```
[bill@home bill]$ man w
```

Figure 7-1 shows the resulting output, which the command displays one page at a time. Notice the colon prompt that appears at the bottom left of the screen. To page forward, press the Space key; to page backward, press the b key. To exit the **man** program, press the q key.

Manpages are organized according to a common format. At the beginning of a manpage, you’ll find the name of the page and the section of the database from which the page comes, shown in parentheses. For example, Figure 7-1 shows “W(1)” in the upper-left and -right corners. This means that you’re looking in section 1 of the manpage (the section pertaining to commands) for the **w** command.

Next in the output comes the name and a brief description of the command. Then comes a synopsis of the command, which shows the options and arguments that you can specify. Brackets enclose parts of a command that you can choose to include or omit. Next comes a detailed description of the operation of the command, followed by a description of its options.

As you’re learning your way around Linux, you may find it convenient to reserve a terminal window or virtual console for running the **man** command. Alternatively, you can browse manpages using Konqueror. Browsing the URI **man://index** will present the Unix manual index, from which you can jump to other manpages. That way, you can enter commands in a separate virtual console, switching between windows or consoles to refresh your recollection of the options and arguments of commands as you type them.
Table 7-4 describes the sections of the manual page database; most sections are primarily of interest to programmers. As a user and system administrator, you’ll be interested primarily in sections 1 and 8.

Table 7-4. Manual page sections

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Executable programs and shell commands</td>
</tr>
<tr>
<td>2</td>
<td>System calls (provided by the kernel)</td>
</tr>
<tr>
<td>3</td>
<td>Library calls (provided by system libraries)</td>
</tr>
<tr>
<td>4</td>
<td>Special files (for example, device files)</td>
</tr>
<tr>
<td>5</td>
<td>File formats and conventions</td>
</tr>
</tbody>
</table>
Using `apropos`

The `man` command searches the manual pages and displays detailed information about a specified command. The `apropos` command also searches the manual pages, but it displays just a one-line summary of each. You have to supply `apropos` with a keyword that indicates the information you’re looking for; it simply displays each summary line that contains the keyword. For example, typing the command:

```
[bill@home bill]$ apropos samba
```

will display a list of manpages containing the word `samba`, as shown in Figure 7-2.

![Output of the apropos command](image)

**Figure 7-2. Output of the apropos command**

The `apropos` command is useful when you don’t recall the name of a Linux command. By typing a related keyword, you can obtain a list of commands and search the list for the command you need.
The `apropos` command uses a special database to store information about commands. Before using the command for the first time, you must create the database by logging in as root and issuing the command:

```
makewhatis
```

The command may require several minutes to complete. After installing a package that includes commands, you can update the database to include information about the new commands by issuing the same command.

### Using Commands That Work with Directories

Now that you understand the fundamentals of issuing Linux commands, you’re ready to learn some commands that work with directories. Rather than simply reading this section, you should log in to your Linux system and try the commands for yourself. By doing so, you will begin to develop skill in working with shell commands.

#### Displaying the working directory

To display the current working directory, issue the `pwd` (print working directory) command. The `pwd` command requires no options or arguments:

```
[bill@home bill]$ pwd
/root
```

The `pwd` command displays the absolute pathname of the current working directory.

#### Changing the working directory

To change the working directory, issue the `cd` (change directory) command, specifying the pathname of the new working directory as an argument. You can use an absolute or relative pathname. For example, to change the working directory to the `/bin` directory, type:

```
[bill@home bill]$ cd /bin
[bill@home /bin]$
```

Notice how the prompt changes to indicate that `/bin` is now the working directory.

You can quickly return to your home directory by issuing the `cd` command without an argument:

```
[bill@home /bin]$ cd
[bill@home bill]$
```

Again, notice how the prompt changes to indicate the new working directory.
If you attempt to change the working directory to a directory that doesn’t exist, Linux displays an error message:

[bill@home bill]$ cd nowhere
bash: nowhere: No such file or directory

Displaying directory contents

To display the contents of a directory, you use the `ls` (list) command. The `ls` command provides many useful options that let you tailor its operation and output to your liking.

The simplest form of the `ls` command takes no options or arguments. It simply lists the contents of the working directory, including files and subdirectories (your own output will differ, reflecting the files present in your working directory):

```
[bill@home bill]$ ls
GNUstep                  firewall                 sniff
Xrootenv.0               linux                    ssh-1.2.26
audio.cddb               mail                     ssh-1.2.26.tar.gz
audio.wav                mirror                   support
axhome                   mirror-2.8.tar.gz        temp
conf                     nlxb3181.tar             test
corel                    openn                    test.doc
drivec.img               scan                     tulip.c
dynip_2.00.tar.gz        screen-3.7.6-0.i386.rpm  win98
```

Here, the output is presented in lexical (dictionary) order, as three columns of data. Notice that filenames beginning with uppercase letters appear before those beginning with lowercase letters.

A more sophisticated form of the `ls` command that includes the `-l` option displays descriptive information along with the filenames, as shown in Figure 7-3.

The first line of the output shows the amount of disk space used by the working directory and its subdirectories, measured in 1 KB blocks. Each remaining line describes a single file or directory. The columns are:

Type

The type of file: a directory (`d`), or an ordinary file (`-`). If your system supports color, Linux displays output lines that pertain to directories in blue and lines that pertain to files in white.

Access modes

The access mode, which determines which users can access the file or directory. You’ll learn more about access modes, links, and groups in subsequent sections of this chapter.

Links

The number of files or directories linked to this one.


Owner
The user who owns the file or directory.

Group
The group that owns the file or directory.

Size
The size of the file or directory, in bytes.

Modification date
The date and time when the file or directory was last modified.

Name
The name of the file or directory.

If a directory contains many files, the listing will fill more than one screen. To view the output one screen at a time, use the following command:

```
[root@desktop /root] # ls -l
```

This command employs the pipe redirector, sending output of the `ls` subcommand to the `less` subcommand, which presents the output one screen at a time. You can control the operation of the `less` command by using the following keys:

- **Space** moves you one page forward.
- **b** moves you one page back.
- **q** or **Q** exits the program and returns you to the command prompt.

If you want to list a directory other than the working directory, you can type the name of the directory as an argument of the `ls` command, like so:

```
ls /bin
```
Linux displays the contents of the directory but does not change the working directory. Similarly, you can display information about a file by typing its name as an argument of the `ls` command. Moreover, the `ls` command accepts an indefinite number of arguments, so you can type a series of directories and filenames as arguments, separating each with one or more spaces.

When the name of a directory or file begins with a dot (.), the output of the `ls` command does not normally include the directory or file, because the file is hidden. To cause the output of the `ls` command to include hidden directories and files, use the `-a` option. For example, to list all the files and subdirectories in the current directory—including hidden ones—type:

```
[bill@home bill]$ ls -a -l
```

If you prefer, you can combine the `-a` and `-l` options, typing the command like this:

```
[bill@home bill]$ ls -al
```

A user’s home directory generally includes several hidden files containing configuration information for various programs. For example, the `.profile` file contains configuration information for the Linux `bash` shell.

The `ls` command provides a host of additional useful options; see its manual page for details.

### Creating a directory

You can create directories by using the `mkdir` (make directory) command. Just type the name of the new directory as an argument of the command. Linux creates the directory as a subdirectory of the working directory. For example, this command creates a subdirectory named `office`:

```
[bill@home bill]$ mkdir office
```

If you don’t want to create the new directory as a subdirectory of the working directory, type an absolute or relative pathname as the argument. For example, to create a directory named `/tmp/documents`, type:

```
[bill@home bill]$ mkdir /tmp/documents
```

The name of a directory or file must follow certain rules. For example, it must not contain a slash (/) character. Directory names and filenames usually include letters (either upper- or lowercase), digits, dots, and underscores (_). You can use other characters, such as spaces and hyphens, but such names present problems, because the shell gives them special meaning. If you simply must use a name containing special characters, enclose the name within single quotes (`'`). The quotes don’t become part of the name that is stored on the disk. This technique is useful when accessing files on a Windows filesystem; otherwise, you’ll have trouble working with files in directories such as `My Documents`, which have pathnames containing spaces.
Most MS-DOS filenames contain a dot, but most Linux filenames do not. In MS-DOS, the dot separates the main part of the filename from a part known as the extension, which denotes the type of the file. For example, the MS-DOS file `memo.txt` would contain text. Most Linux programs ignore file extensions, so Linux filenames don’t require an extension. However, if you plan to send a file to someone using an operating system other than Linux, you should include an appropriate file extension, such as `.txt` for a text file.

Removing a directory

To remove a directory, use the `rmdir` command. For example, to remove `unwanted`, a subdirectory of the working directory, type:

```
[bill@home bill] $ rmdir unwanted
```

If the directory you want to delete is not a subdirectory of the working directory, remove it by typing an absolute or relative pathname.

You cannot use `rmdir` to remove a directory that contains files or subdirectories; you must first delete the files in the directory and then remove the directory itself. Hidden files can present a puzzle, because a directory containing hidden files may look empty even though it’s not. You must delete hidden files before you can delete the directory containing them.

Working with Files

Directories contain files and other directories. You use files to store data. This section introduces you to several useful commands for working with files.

Displaying the contents of a file

Linux files, like most Windows files, can contain text or binary information. The contents of a binary file are meaningful only to skilled programmers, but you can easily view the contents of a text file. Simply type the `cat` command, specifying the name of the text file as an argument. For example:

```
[root@desktop /root]# cat /etc/passwd
```

displays the contents of the `/etc/passwd` file, which lists the accounts on the system.

If a file is too large to be displayed on a single screen, the first part of the file will whiz past you and you’ll see only the last few screenfulls of lines of the file. To avoid this, you can use the `less` command:

```
[root@desktop /root]# less /etc/passwd
```

This command displays the contents of a file in the same way the `man` command displays a manual page. You can use `Space` and the `b` key to page forward and backward through the file, and the `q` or `Q` key to exit the command.
Removing a file
To delete a file, type the `rm` (remove) command, specifying the name of the file as an argument. For example:

```
[bill@home bill]$ rm badfile
```
removes the file named `badfile` contained in the working directory. If a file is located elsewhere, you can remove it by specifying an absolute or relative pathname.

Once you remove a Linux file, its contents are likely lost forever. Be careful to avoid removing a file that contains needed information. Better yet, be sure to have a backup copy of any important data.

The `-i` option causes the `rm` command to prompt you to verify your decision to remove a file. If you don’t trust your typing skills, you may find this option helpful. If you log in as the root user, Linux automatically supplies the `-i` option even if you don’t type it.

Copying a file
To copy a file, use the `cp` command, specifying the name (or path) of the file you want to copy and the name (or path) to which you want to copy it. For example:

```
[root@desktop /root]# cp /etc/passwd sample
```
copies the `/etc/passwd` file to a file named `sample` in the working directory.

If the destination file already exists, Linux overwrites it. You must therefore be careful to avoid overwriting a file that contains needed data. Before copying a file, use the `ls` command to ensure that no file will be overwritten; alternatively, use the `-i` option of the `cp` command, which prompts you to verify that you want to overwrite an existing file. If you log in as the root user, Linux automatically supplies the `-i` option even if you don’t type it.

Renaming or moving a file
To rename a file, use the `mv` command, specifying the name (or path) of the file and the new name (or path). For example:

```
[bill@home bill]$ mv old new
```
renames the file named `old` as `new`. If the destination file already exists, Linux overwrites it, so you must be careful. Before moving a file, use the `ls` command to ensure that no file will be overwritten or use the `-i` option of the `mv` command, which prompts you to verify that you want to overwrite an existing file. If you log in as the root user, Linux automatically supplies the `-i` option even if you don’t type it.

The `mv` command can rename a directory but sometimes cannot move a directory from one device to another. If you encounter problems when moving a directory to a new device, first copy the directory and its contents and then remove the original.
Finding a file

If you know the name of a file but do not know which directory contains it, you can use the `find` command to locate the file. For example:

```
[bill@home bill]$ find . -name 'missing' -print
```

attempts to find a file named `missing`, located in (or beneath) the current working directory (`.`). If the command finds the file, it displays its absolute pathname.

If you know only part of the filename, you can surround the part you know with asterisks (`*`):

```
[bill@home bill]$ find / -name '*iss*' -print
```

This command will find any file whose name includes the characters `iss`, searching every subdirectory of the root directory (that is, the entire system).

Another command useful for finding files is `locate`. The `locate` command uses a database that is updated only daily. So it can’t find recently created files and it shows files that may have been recently deleted. But it operates much more quickly than the `find` command. To use the `locate` command, specify as the command’s argument a string of characters, which need not be enclosed in quotes. The command will list all filenames in its database that contain the specified characters. For example, the command:

```
locate pass
```

lists all files containing the characters `pass`.

The `locate` command depends on a database built by the `updatedb` command. You must login as root and run `updatedb` before running `locate` for the first time and as often thereafter as you feel that your system’s file structure has changed significantly. The `cron` and `anacron` services will generally take care of this for you. However, you can run the `updatedb` command manually if the results of the `locate` command seem out of date.

Printing a file

If your system includes a configured printer, you can print a file by using the `lpr` command. For example:

```
[root@desktop /root]# lpr /etc/passwd
```

`/etc/passwd` to the printer. See Chapter 9 for information on configuring a printer.

You can send other files to the printer while a file is printing. The `lpq` command lets you see what files are queued to be printed:

```
[root@desktop /root]# lpq
lp is ready and printing
Rank Owner Job Files Total Size
active root 155 /etc/passwd 1030 bytes
```

---

Working with the Linux Command Prompt | 141
Each waiting or active file has an assigned print job number. You can use `lprm` to cancel printing of a file, by specifying the print job number. For example:

```
[root@desktop /root]# lprm 155
```
cancels printing of job number 155. However, only the user who requested that a file be printed (or the root user) can cancel printing of the file.

**Working with compressed files**

To save disk space and expedite downloads, you can compress a data file. By convention, compressed files are named ending in `.gz`; however, Linux doesn’t require or enforce this convention.

To expand a compressed file, use the `gunzip` command. For example, suppose the file `bigfile.gz` has been compressed. Typing the command:

```
[bill@home bill]$ gunzip bigfile.gz
```
extracts the file `bigfile` and removes the file `bigfile.gz`.

To compress a file, use the `gzip` command. For example, to compress the file `bigfile`, type the command:

```
[bill@home bill]$ gzip bigfile
```
The command creates the file `bigfile.gz` and removes the file `bigfile`.

A different compression format is provided by the `bzip2` command; files compressed this way usually have `.bz2` extensions and can be uncompressed with `bunzip2`.

Sometimes it’s convenient to store several files (or the contents of several subdirectories) in a single file. This is useful, for example, in creating a backup or archive copy of files. The Linux `tar` command creates a single file that contains data from several files. Unlike the `gzip` command, the `tar` command doesn’t disturb the original files.

To create a tarfile, as a file created by the `tar` command is called, issue a command like this:

```
tar -cvf tarfile files-or-directories
```
Substitute `tarfile` with the name of the tarfile you want to create and `files-or-directories` with a list of files and directories, separating the list elements by one or more spaces. You can use absolute or relative pathnames to specify the files or directories. By convention, the name of a tarfile ends with `.tar`, but Linux does not require or enforce this convention. Some people refer to tarfiles as `tarballs`, because they often contain multiple files.

For example, to create a tarfile named `backup.tar` that contains all the files in all subdirectories of the directory `/home/bill`, type:

```
[bill@home bill]$ tar -cvf backup.tar /home/bill
```
The command creates the file `backup.tar` in the current working directory.
You can list the contents of a tarfile by using a command that follows this pattern:

```
    tar -tvf tarfile | less
```

The `| less` causes the output to be sent to the `less` command, so that you can page through multiple pages. If the tarfile holds only a few files, you can omit `| less`.

To extract the contents of a tarfile, use a command that follows this pattern:

```
    tar -xvf tarfile
```

This command expands the files and directories contained within the tarfile as files and subdirectories of the working directory.

If a file or subdirectory already exists, the `tar` command silently overwrites it.

The `tar` command provides a host of useful options; see its manpage for details.

It’s common to compress a tarfile, which you can do by specifying the options `-czvf` instead of `-cvf`. Compressed tarfiles are conventionally named ending with `.tgz`. To expand a compressed tarfile, specify the options `-xzvf` instead of `-xvf`.

The `tar` command doesn’t use the ZIP method of compression common in the Windows world. However, Linux can easily work with, or even create, ZIP files.

To create a ZIP file that holds compressed files or directories, issue a command like this one:

```
    zip -r zipfile files_to_zip
```

where `zipfile` names the ZIP file that will be created and `files_to_zip` specifies the files and directories to be included in the ZIP file.

To expand an existing ZIP file, issue a command like this one:

```
    unzip zipfile
```

**Working with links**

Windows supports shortcuts, which let you refer to a file or directory (folder) by several names. Shortcuts also let you include a file in several directories or a subdirectory within multiple parent directories. In Linux, you accomplish these results by using the `ln` command, which links multiple names to a single file or directory. These names are called *symbolic links*, *soft links*, *symlinks*, or simply *links*.

To link a new name to an existing file or directory, type a command that follows this pattern:

```
    ln -s old new
```
For example, suppose that the current working directory contains the file william. To be able to refer to this same file by the alternative name bill, type the command:

```
[bill@home bill]$ ln -s william bill
```

The `ls` command shows the result:

```
[bill@home bill]$ ls -l
lrwxrwxrwx  1 root     root     7 Feb 27 13:58 bill->william
-rw-r--r--  1 root     root  1030 Feb 27 13:26 william
```

The new file (bill) has type 1, which indicates it’s a link rather than a file or directory. Moreover, the `ls` command helpfully shows the name of the file to which the link refers (william). Notice the file size of the link bill. Creating a link merely creates a pointer to a file rather than a duplicate of the file, thereby saving disk space.

If you omit the `-s` option, Linux creates what’s called a hard link. A hard link must be stored on the same filesystem as the file to which it refers, a restriction that does not apply to symbolic links. The link count displayed by the `ls` command reflects only hard links; symbolic links are ignored. Hard links are seldom used, because soft links are more flexible.

### Working with file permissions

As explained in Chapter 4, access permissions determine which operations a user can perform on a directory or file. Table 7-5 lists the possible permissions and explains the meaning of each. Recall from Chapter 4 that permissions work differently for directories than for files. For example, permission `r` denotes the ability to list the contents of a directory or read the contents of a file. A directory or file can have more than one permission. Only the listed permissions are granted; any other operations are prohibited. For example, a user who had file permission `rw` could read or write the file but could not execute it, as indicated by the absence of the execute permission, `x`. Look back to Figure 7-3 to see how the `ls` command displays permissions.

**Table 7-5. Access permissions**

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning for a directory</th>
<th>Meaning for a file</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>List the directory</td>
<td>Read contents</td>
</tr>
<tr>
<td>w</td>
<td>Create or remove files</td>
<td>Write contents</td>
</tr>
<tr>
<td>x</td>
<td>Access files and subdirectories</td>
<td>Execute</td>
</tr>
</tbody>
</table>

The access modes of a directory or file consist of three permissions:

**User/Owner**

Applies to the owner of the file

**Group**

Applies to users who are members of the group assigned to the file

**Other**

Applies to other users
The `ls` command lists the file access modes in the second column of its long output format, as shown in Figure 7-3. The column contains nine characters: the first three specify the access allowed the owner of the directory or file, the second three specify the access allowed users in the same group as the directory or file, and the final three specify the access allowed to other users (see Figure 7-4).

![Access modes diagram](image)

**Figure 7-4. Access modes specify three permissions**

You set the access modes of a directory or file by using the `chmod` command, which has the following pattern:

```
chmod nnn directory-or-file
```

The argument `nnn` is a three-digit number, which gives the access mode for the owner, group, and other users. Table 7-6 shows each possible digit and the equivalent access permission. For example, the argument `751` is equivalent to `rwxr-xr-x`, which gives the owner every possible permission, gives the group read and execute permission, and gives other users execute permission.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>--x</td>
</tr>
<tr>
<td>2</td>
<td>-w-</td>
</tr>
<tr>
<td>3</td>
<td>-wx</td>
</tr>
<tr>
<td>4</td>
<td>r--</td>
</tr>
<tr>
<td>5</td>
<td>r-x</td>
</tr>
<tr>
<td>6</td>
<td>rw-</td>
</tr>
<tr>
<td>7</td>
<td>rwx</td>
</tr>
</tbody>
</table>

Table 7-6. Numerical access mode values

If you’re the owner of a file or directory (or if you’re the root user), you can change the ownership of the file or directory by using the `chown` command. For example, the following command assigns `newuser` as the owner of the file `hotpotato`:

```
[chown@home bill]$ chown newuser hotpotato
```

The owner of a file or directory (and the root user) can also change the group of a file. For example, the following command assigns `newgroup` as the new group of the file `hotpotato`:

```
[chown@home bill]$ chgrp newgroup hotpotato
```
The group you assign to a file or directory must have been previously established by the root user. And, unless the command is issued by the root user, the user must be a member of the new group to which the file is assigned. The valid groups appear in the file /etc/group, which only the root user can alter. The root user can assign each user to one or more groups. When you log on to the system, you are assigned to one of these groups—your login group—by default. To change to another of your assigned groups, you can use the `newgrp` command. For example, to change to the group named secondgroup, use the following command:

```
[root@desktop /root]# newgrp secondgroup
```

If you attempt to change to a group that does not exist or to which you have not been assigned, your command will fail. When you create a file or directory, it is automatically assigned your current group as its owning group.

### Running programs

In Linux, as in MS-DOS and Windows, programs are stored in files. Often, you can launch a program by simply typing its filename. However, this assumes that the file is stored in one of a series of directories known as the *path*. A directory included in this series is said to be *in the path* or *on the path*. If you’ve worked with MS-DOS, you’re familiar with the MS-DOS path, and the Linux path works much like it.

If the file you want to launch is not stored in a directory on the path, you can simply type the pathname of the file. Linux will then launch the program even though it’s not on the path. If the file you want to launch is stored in the current working directory, type `./` followed by the name of the program file. Again, Linux will launch the program even though it’s not on the path.

For example, suppose the program `bigdeal` is stored in the directory `/home/bob`, which is the current directory and which happens to be on the path. You could launch the program with any of these commands:

```
bigdeal
./bigdeal
/home/bob/bigdeal
```

The first command assumes that the program is on the path. The second assumes that the program resides in the current working directory. The third explicitly specifies the location of the file.

### Mounting and Unmounting Drives

To mount a device or partition, you use the `mount` command, which has the following pattern:

```
mount options device directory
```
The `mount` command provides many options. However, you can generally use the `mount` command without any options; consult `mount`'s manpage to learn about the available options.

The reason you can often use the `mount` command without options is that the file `/etc/fstab` describes your system’s devices and the type of filesystem each is likely to contain. When you reboot your system after adding a new device, the `kudzu` service detects the new device and updates `/etc/fstab` to include an entry for the device, if necessary. However, you can edit the `/etc/fstab` by hand to add new entries or revise entries placed there by `kudzu`.

You must specify the device that you want to mount and a directory, known as the `mount point`. To make it convenient to access various devices, Linux treats a mounted device as a directory; mounting the device associates it with the named directory. For example, the following command is used to mount a CD-ROM:

```
[root@desktop /root]# mount -t iso9660 /dev/cdrom /mnt/cdrom -o ro
```

The file `/dev/cdrom` is a link that points to the actual device file associated with your system’s CD-ROM drive. The directory `/mnt/cdrom` is a directory created by the install program; this directory is conventionally used as the mounting point for CD-ROMs. The type of filesystem found on most CD-ROMs is `iso9660`, the value of the `-t` argument. The `-o` argument, `ro`, specifies that the filesystem is read-only; that is, it can be read but not written. The file `/etc/fstab` can supply most of these arguments if they’re omitted. Generally, you can mount a CD-ROM by issuing the abbreviated command:

```
[root@desktop /root]# mount /dev/cdrom
```

After the command has completed, you can access files and directories on the CD-ROM just as you would access ordinary files and directories on the path `/mnt/cdrom`. For example, to list the top-level files and directories of the CD-ROM, simply type:

```
[root@desktop /root]# ls /mnt/cdrom
```

To mount an MS-DOS floppy disk in your `a:` drive, type:

```
[root@desktop /root]# mount -t msdos /dev/fd0 /mnt/floppy
```

To unmount a device, specify its mount point as an argument of the `umount` command (note the missing `n` in `umount`). For example, to unmount a CD-ROM diskette, type:

```
[root@desktop /root]# umount /mnt/cdrom
```

Generally, only the root user can unmount a device. However, Red Hat Linux allows ordinary users to mount and unmount devices when logged in locally. Nevertheless, a device can be unmounted only if it’s not in use. For example, if a user’s working directory is a directory of the device, the device cannot be unmounted.
If you can’t unmount a device, check each terminal window and virtual console to see if one of them has a session that’s using the device as its working directory. If so, either exit the session or change to a working directory that isn’t associated with the device.

**Formatting a Floppy Disk**

Before you can write data on a floppy disk, you must format it. The Linux command to format a floppy disk is `fdformat`. Simply follow the command with an argument that specifies the floppy drive and the capacity of the floppy disk; the available arguments are listed in Table 7-7.

Table 7-7. Floppy drive designators

<table>
<thead>
<tr>
<th>Designation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/fd0</td>
<td>3.5-inch disk in a: (1.44 MB)</td>
</tr>
<tr>
<td>/dev/fd0H1440</td>
<td>3.5-inch disk in a: (1.44 MB)</td>
</tr>
<tr>
<td>/dev/fd1</td>
<td>3.5-inch disk in b: (1.44 MB)</td>
</tr>
<tr>
<td>/dev/fd1H1440</td>
<td>3.5-inch disk in b: (1.44 MB)</td>
</tr>
<tr>
<td>/dev/fd1H2880</td>
<td>3.5-inch disk in b: (2.88 MB)</td>
</tr>
</tbody>
</table>

For example, to format a 1.44 MB floppy disk, log in as `root` and issue the command:

```
[root@desktop /root]# fdformat /dev/fd0H1440
```

The `fdformat` command performs only a low-level format. Before the floppy disk can be used, you must place a filesystem on it. Floppy disks containing an MS-DOS filesystem are useful for transferring data between Windows and Linux. To place an MS-DOS filesystem on a formatted floppy disk, issue the command:

```
[root@desktop /root]# mkdosfs /dev/fd0
```

Once the floppy disk has been formatted and given a filesystem, you can mount it and then read and write it.

Be sure you unmount the floppy disk before you remove it. Unmounting the floppy disk ensures that all pending data has been written to it; otherwise, the floppy disk may be unusable due to corrupt data.

**Useful Linux Programs**

This section presents several programs you may find helpful in working with your Linux system. You’ll learn several commands that report system status and you’ll learn how to use `nano`, a simple text editor.
Viewing System Information

Linux provides a number of commands that report system status. The most commonly used commands are shown in Table 7-8. These commands can help you troubleshoot system problems and identify resource bottlenecks. Although each command can be used without options or arguments, each supports options and arguments that let you customize operation and output; consult the appropriate manpage for details.

Table 7-8. Useful system commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>df</td>
<td>Shows the amount of free disk space (in 1 KB blocks) on each mounted filesystem.</td>
</tr>
<tr>
<td>du</td>
<td>Shows the amount of disk space (in 1 KB blocks) used by the working directory and its subdirectories. With the -s option, displays just a summary without listing all the subdirectories and files.</td>
</tr>
<tr>
<td>free</td>
<td>Shows memory usage statistics, including total free memory, memory used, physical memory, swap memory, shared memory, and buffers used by the kernel.</td>
</tr>
<tr>
<td>ps</td>
<td>Shows the active processes (instances of running programs) associated with this login session. Use the -a option to list all processes.</td>
</tr>
<tr>
<td>top</td>
<td>Shows a continually updated display of active processes, and the resources they are using. Type the q key to exit.</td>
</tr>
<tr>
<td>uptime</td>
<td>Shows the current time, the amount of time logged in, the number of users logged in, and system load averages.</td>
</tr>
<tr>
<td>users</td>
<td>Shows each login session.</td>
</tr>
<tr>
<td>w</td>
<td>Shows a summary of system usage, currently logged-in users, and active processes.</td>
</tr>
<tr>
<td>who</td>
<td>Shows the names of users currently logged in, the terminal each is using, the time each has been logged in, and the name of the host from which each logged in (if any).</td>
</tr>
</tbody>
</table>

Using the nano Editor

If you’re working under X, you have access to a variety of GUI text editors. However, GUI text editors cannot be used from a virtual console. The nano editor is a simple text editor that you can think of as the Linux equivalent of the Windows program named Edit, because it can be used in graphical or text mode.

To start nano, simply type nano at the shell prompt, or if you want to create or edit a particular file, type nano followed by the name of the file (or the file’s path, if the file is not in the working directory). For example, to create or edit the file mydata, type:

```
[bill@home bill]$ nano mydata
```

Figure 7-5 shows nano’s standard display. At the top of the display is a status line, which shows the version of the program and the name of the file being edited (or New Buffer, if the file is new). If the file has been modified, the upper-right corner of the display contains the word Modified. The bottom two lines of the display list the available editing commands. Most of the commands require you to type a control
character, so that commands can be distinguished from characters you want to add to the buffer, as nano’s work area is termed. Typing an ordinary character inserts it at the current cursor position. You can use the cursor keys to move around the display; you can use the Delete or Backspace key to erase unwanted characters. Some commands use the third line from the bottom to report status and obtain additional input.

Table 7-9 summarizes nano’s commands. Notice that the command Ctrl-G accesses nano’s help system. You can access several of the commands by using function keys; for example, pressing F1 has the same result as typing Ctrl-G.

Table 7-9: Summary of nano commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl-^</td>
<td>Mark the cursor position as beginning of selected text.</td>
</tr>
<tr>
<td>Ctrl-A</td>
<td>Move to the beginning of the current line.</td>
</tr>
<tr>
<td>Ctrl-B</td>
<td>Move backward one character.</td>
</tr>
<tr>
<td>Ctrl-C (F11)</td>
<td>Report the current cursor position.</td>
</tr>
<tr>
<td>Ctrl-D</td>
<td>Delete the character at the cursor position.</td>
</tr>
<tr>
<td>Ctrl-E</td>
<td>Move to the end of the current line.</td>
</tr>
<tr>
<td>Ctrl-F</td>
<td>Move forward one character.</td>
</tr>
<tr>
<td>Ctrl-G (F1)</td>
<td>Display help.</td>
</tr>
<tr>
<td>Ctrl-I</td>
<td>Insert a tab at the current cursor position.</td>
</tr>
<tr>
<td>Ctrl-J (F4)</td>
<td>Format the current paragraph.</td>
</tr>
<tr>
<td>Ctrl-K (F9)</td>
<td>Cut selected text.</td>
</tr>
<tr>
<td>Ctrl-L</td>
<td>Refresh the display.</td>
</tr>
<tr>
<td>Ctrl-N</td>
<td>Move to the next line.</td>
</tr>
<tr>
<td>Ctrl-O (F3)</td>
<td>Save the current buffer to a file.</td>
</tr>
<tr>
<td>Ctrl-P</td>
<td>Move to the previous line.</td>
</tr>
<tr>
<td>Ctrl-R (F5)</td>
<td>Insert an external file at the current cursor position.</td>
</tr>
<tr>
<td>Ctrl-T (F12)</td>
<td>Invoke the spelling checker.</td>
</tr>
</tbody>
</table>
Here’s a simple exercise that will give you a quick tour of nano. Start nano by issuing the command:

```
nano
```

Then type the following short paragraph of text, including the typographical errors:

```
Nano is is a greet editor. I use it for my light-duty tasks editing. However, when I need to really get down to business, I prefer vi.
```

Notice that three errors appear:

- The word is appears twice
- Great is misspelled as greet
- The words editing and tasks appear in the wrong order.

To correct the first error, use the up and left arrow keys to position the cursor on the letter i of the first instance of the word is. Then, press Delete three times to erase the word and the following space.

Next, use the right arrow key to place the cursor on the second (incorrect) letter e in the misspelled word greet. Type an a and press Delete to correct the error by replacing the e with an a.

Now, let’s revise the phrase “for my” to read “for all my.” Simply use the right arrow keys to move to the letter m in the word my. Type all and a space.

Finally, let’s cut and paste to move the words tasks and editing into their proper sequence. Use the arrow keys to position the cursor on the letter t in the word tasks. Type Ctrl-A to mark the beginning of a selection. Use the right arrow key to select the remainder of the word and the following text, positioning the cursor on the letter e in the word editing. Cut the selected word by typing Ctrl-K. Now, use the right arrow key to move the cursor past the word editing and type Ctrl-U to paste the text that was cut. The paragraph is now error-free.

Save the paragraph to a file by typing Ctrl-X, responding y to the question “Save modified buffer?”, typing the desired filename, and pressing Enter. Use the less command to verify that the file was created and has the proper contents.