4. Type **pwd** to view your current directory. It will probably be `/home/yourusername`, where *yourusername* is—you guessed it!—your username.
5. Type **`cat ../../etc/fstab`** to view this configuration file using a relative file reference. The first `..` in this command refers to `/home`, and the second refers to the root (`/`) directory. (If your home directory is in an unusual location, you may need to adjust the number of `../` elements in this command, which is why I had you use `pwd` to find your current directory in the previous step.)
6. Type **`cat ~/../../etc/fstab`** to view this configuration file using a home directory reference.

Of course, steps 5 and 6 use rather awkward file references; in real life, you'd probably use an absolute file reference to access `/etc/fstab` from your home directory. If you were in a subdirectory of `/etc`, though, typing `../fstab` would be slightly easier than typing `/etc/fstab`; and typing `~/afile.txt` would be easier than typing the complete path to your home directory.

Using Common File Manipulation Commands

Chapter 7, "Managing Files," describes the most common commands used to manipulate files in detail, and Chapter 15 describes commands related to file ownership and permissions. Some of these commands are used in the remainder of this chapter, so Table 6.2 summarizes them.

Remember `man`! You can use it to learn about most Linux commands and utilities, including the common file manipulation commands.

TABLE 6.2 Common file manipulation commands

Command	Effect
`cat`	Displays files on standard output. Two or more files can be specified and output redirected to merge them together.
`chgrp`	Changes group ownership of a file. Described in more detail in Chapter 15.
`chmod`	Changes permissions of a file. Described in more detail in Chapter 15.
`chown`	Changes ownership of a file. Described in more detail in Chapter 15.
`cp`	Copies a file. Described in more detail in Chapter 7.

(Continues)

TABLE 6.2 *(Continued)*

Command	Effect
`echo`	Echoes the text you enter on the screen. Although this isn't technically a file command, it can be used with redirection to create or add text to a file.
`head`	Displays the first few lines of a text file.
`less`	Displays a file a page at a time.
`ln`	Creates links to files. Described in more detail in Chapter 7.
`ls`	Displays files in a directory, as described earlier.
`mkdir`	Creates a new directory. Described in more detail in Chapter 7.
`mv`	Moves or renames a file. Described in more detail in Chapter 7.
`pwd`	Prints the name of the current working directory.
`rm`	Removes a file. Described in more detail in Chapter 7.
`rmdir`	Removes a directory. Described in more detail in Chapter 7.
`tail`	Displays the last few lines of a text file.
`wc`	Counts characters, words, and lines in a text file. Described in more detail in Chapter 10.

Using Shell Features

Bash includes several features that make using it much easier. I've already described some of these. Many others are beyond the scope of this book. Two, however, deserve attention even in a brief introduction to shells: command completion and command history.

Using Command Completion

Command completion is the hero of everybody who hates typing: It's a way to enter a long command or filename with a minimal number of keystrokes. To

Some of the details of how command completion works vary from one distribution to another.

use command completion, you type part of a command or filename and then press the Tab key. If only one command on the path completes the command, Bash fills in the rest—and likewise when using command completion to refer to files. To illustrate the use of command completion, you can try it out with a few commands:

1. Launch a shell.
2. Type **`if`** followed by pressing the Tab key. The computer will probably beep or sound a tone. This indicates that your incomplete command could be completed by multiple commands, so you must type more characters. (In some configurations, the computer skips straight to the next step, as if you'd pressed Tab twice.)
3. Press the Tab key again. The shell displays a list of possible completions, such as `if`, `ifconfig`, and `ifdown`.
4. Type **`co`**, making your command so far **`ifco`**, and press the Tab key again. The computer will probably complete the command: **`ifconfig`**. (If it doesn't, another program that completes the command may exist on your computer, so you may need to type another character or two.)
5. Press the Enter key. The computer runs `ifconfig`, which displays information on your network connections.

Chapter 17, "Managing Network Connections," describes `ifconfig` in more detail.

Sometimes, command completion will be able to partially complete a command. For instance, typing **`gru`** and then pressing Tab is likely to add a single unique character, `b`; however, several commands begin with `grub`, so you must then add more characters yourself. (These commands deal with the Grand Unified Bootloader, GRUB, which helps Linux to boot.)

Command completion also works with files. For instance, you can type **`cat /etc/ser`** followed by the Tab key to have Bash complete the filename, and therefore the command, as **`cat /etc/services`**. (This command shows you the contents of a Linux configuration file.)

Using Command History

Bash remembers the recent commands you've typed, and you can use this fact to save yourself some effort if you need to type a command that's similar to one you've typed recently. In its most basic form, you can use the up arrow key to enter the previous command; pressing the up arrow repeatedly moves backward

through earlier and earlier commands. Table 6.3 summarizes some other commonly used keystrokes you can use in the command history—or even when editing new commands.

TABLE 6.3 Bash editing and command history features

Keystroke	Effect
Up arrow	Retrieves the previous entry from the command history.
Left arrow	Moves the cursor left one character.
Right arrow	Moves the cursor right one character.
Ctrl+A	Moves the cursor to the start of the line.
Ctrl+E	Moves the cursor to the end of the line.
Delete key	Deletes the character under the cursor.
Backspace key	Deletes the character to the left of the cursor.
Ctrl+T	Swaps the character under the cursor with the one to the left of the cursor.
Ctrl+X then Ctrl+E	Launches a full-fledged editor on the current command line.
Ctrl+R	Searches for a command. Type a few characters and the shell will locate the latest command to include those characters. You can search for the next-most-recent command to include those characters by pressing Ctrl+R again.

Many of the Bash command editing features are similar to those used by the `emacs` text editor.

As an example of command history in use, try this:

1. Type **`cd /tmp`** to change to the `/tmp` directory, in which many programs store temporary files.
2. Type **`ls`** to see a list of the files in your current directory (`/tmp`).
3. Press the up arrow key. Your `ls` command should re-appear.
4. Press the spacebar and type in ~ to make the new command **`ls ~`**, and then press Enter. You should now see the contents of your home directory.

5. Type **`cat /etc/fstab`** to see the contents of `/etc/fstab`, which is a file that defines how disk space is used.
6. Press Ctrl+R. The Bash prompt will change to read `(reverse-i-search)`':`.
7. Type **`l`** (without pressing Enter). Your earlier **`ls ~`** command will appear.
8. Press Ctrl+R again. The command should change to a simple **`ls`**—the one you entered in step 2.
9. Press Enter. The **`ls`** command should execute again.

The Ctrl+R search feature searches on anything you enter on a command line—a command name, a filename, or other command parameters.

Another history feature is the `history` command. Type **`history`** to view all the commands in your history, or add a number (as in **`history 10`**) to view the most recent specified number of commands.

I encourage you to experiment with these features. Tab completion and command history are both powerful tools that can help you avoid a great deal of repetitive typing. Command history can also be a useful memory aid—if you've forgotten the exact name of a file or command you used recently, you might be able to retrieve it by searching on part of the name you *do* remember.

The Essentials and Beyond

Command lines are powerful tools in Linux; they're the basis on which many of the friendlier GUI tools are built, they can be accessed without the help of a GUI, and they can be scripted. To use the text-mode tools described in other chapters of this book, you should be familiar with the basics of a Linux shell. These include knowing how to start a shell, how to run programs in a shell, how to manipulate files, and how to use a shell's time-saving features.

Suggested Exercises

- Read the man pages for the following commands: `man`, `less`, `cat`, `cd`, `ls`, `grep`, and `su`.
- Launch a GUI program, such as `gedit`, with and without a trailing ampersand (&). When you launch it without an ampersand, use Ctrl+Z to put it into the background and see how the program reacts to mouse clicks. Use `fg` to return it to the foreground, then repeat the process but use `bg` to run the program in the background. See what happens in your terminal when you exit from the GUI program.

(Continues)

The Essentials and Beyond *(Continued)*

- In a shell, type a single letter, such as **m**, and press the Tab key. What happens? What happens if you type a less common letter, such as **z**, and then press Tab?
- Experiment with the command history. Use it to search on strings that are part of both command names and filenames you've used. Use the arrow keys and editing features described in Table 6.3 to edit commands you've used previously.

Review Questions

1. What keystroke moves the cursor to the start of the line when typing a command in Bash?

   A. Ctrl+A
   B. Left arrow
   C. Ctrl+T
   D. Up arrow
   E. Ctrl+E

2. How can you run a program in the background when launching it from a shell? (Select all that apply.)

   A. Launch the program by typing **`start` *`command`***, where *command* is the command you want to run.
   B. Launch the program by typing **`bg` *`command`***, where *command* is the command you want to run.
   C. Append an ampersand (&) to the end of the command line.
   D. Launch the program normally, type Ctrl+Z in the shell, and then type **`bg`** in the shell.
   E. Launch the program normally, type Ctrl+Z in the shell, and then type **`fg`** in the shell.

3. Which of the following commands, typed at a Bash prompt, returns you to your home directory?

   A. **`home`**
   B. **`cd /home`**
   C. **`cd homedir`**
   D. **`homedir`**
   E. **`cd ~`**

4. True or false: The Alt+F2 keystroke, typed in X, brings up a text-mode display you can use to log into Linux.

5. True or false: The filename `..\upone.txt` refers to the file `upone.txt` in the parent of the current directory.

(Continues)

The Essentials and Beyond *(Continued)*

6. True or false: The `-r` option to `ls` creates a recursive directory listing.
7. The _____ command displays the path to the current working directory.
8. To view all files, including hidden files and directories, in the current directory, you would type **`ls`** _____.
9. The _____ command displays text files or can concatenate multiple files together.

CHAPTER 7

Managing Files

Much of what you do with a computer involves manipulating files. Most obviously, files hold the correspondence, spreadsheets, digital photos, and other documents you create. Files also hold the configuration settings for Linux—information on how to treat the network interfaces, how to access hard disks, and what to do as the computer starts up. Indeed, even access to most hardware devices and kernel settings is ultimately done through files. Thus, knowing how to manage these files is critically important for administering a Linux computer. This chapter begins with a description of the basic text-mode commands for manipulating files. Directories are files, too, so this chapter covers directories, including the commands you can use to create and manipulate them.

- **Manipulating files**
- **Manipulating directories**

Manipulating Files

If you've used Windows or Mac OS X, chances are you've used a GUI file manager to manipulate files. Such tools are available in Linux, as noted in Chapter 4, "Using Common Linux Programs," and you can certainly use a file manager for many common tasks. Linux's text-mode shells, such as Bash, provide simple but powerful tools for manipulating files, too. These tools can simplify some tasks, such as working with all the files with names that include the string `invoice`. Thus, you should be familiar with these text-mode commands.

To begin this task, I describe some ways you can create files. With files created, you can copy them from one location to another. You may sometimes want to move or rename files, so I explain how to do so. Linux enables you to create *links*, which are ways to refer to the same file by multiple names. If you never want to use a file again, you can delete it. *Wildcards* provide the means to refer to many files using a compact notation, so I describe them. Finally, I cover the case-sensitive nature of Linux's file manipulation commands.