Chapter 1

A Linux Primer

If you are serious about computing and understanding how an operating system works, Linux is the system of choice. To learn Linux you must use it, and, of course, to use it you must learn it. Such a paradox is rather common—you probably learned to drive a car this way. You just need some basic help and pointers to get started. Here we present an overview of basics. Once you understand the material in this chapter, you will be able to use the operating system to learn more in each successive chapter. At first, you need a learner's permit to drive a car. Consider this chapter your learner's permit for Linux; with a little practice you will be using Linux almost right away.

Learning Linux involves understanding how to use it from the user level and how to program it from the system level. This primer provides basic information and a selection of topics designed to get you started using Linux quickly. As you read this chapter, try the different commands and features as you come to them. In each case, you are given enough information to get you on the system and learning.

1.1 What Is an Operating System?

The operating system controls a computer and makes it usable. It brings life to the innate electronic hardware components and orchestrates all activities on a computer. The same hardware under a different operating system is literally a different computer.

The operating system provides service and control functions to users, programs, files, operators, display monitors, printers, network connections, and everything else on a computer system. A computer operating is one of the most complicated and sophisticated objects humans ever built.

A modern operating system like Linux consists of two parts: a *kernel* and a set of commands and applications. The kernel deals with central functions, including concurrent program execution, memory management, input/output (I/O), file services, and network interface. Commands and applications supply other operations such as Shells, language compilers, text editors, email processors, Web browsers, software package managers, sound and video tools, and so on.

A *Shell* is a special program that is very important in Linux. It is a com-

mand interpreter that allows users to type commands and run programs. We will have much to say about the Shell.

1.2 Getting Started: Login and Logout

To access your Linux system, you must have a user account, identified by a *userid* and a *password*, that have been created by a system administrator. At most installations, your userid will be your last name or your first initials and last name (often all lowercase).

Your password is a safeguard against unauthorized use of your computer account. You need to choose a password of at least eight characters (your local system may enforce other conventions as well, such as a minimum length or that there be at least one numeral or symbol). Correctly spelled words or names of relatives are bad choices for passwords. A sequence containing upper and lower case characters, digits, symbols, and control characters is usually better. Since you are the only one who knows your password, you must be careful with it. Forgetting your password means the system administrator must create a new one for you. Giving your password to the wrong person could have even more dire consequences; you could be blamed for whatever damage is caused, intentionally or otherwise, by the other person. The best rule is do not tell anybody your password, but keep it written down somewhere safe.

Once you have a userid and password, you can begin your Linux session. The first step is the login procedure. Login protects the system against unauthorized use and authenticates the identity of the user. You can use Linux from the *console* or across a network.

FIGURE 1.1: Linux Login Screen

Desktop Login

Find a computer displaying the Linux *desktop login screen* (Figure 1.1). This can be the *console* where the keyboard, mouse, and display are directly con-

nected to the computer hardware running the Linux system. Or it can be a different computer on the LAN (Local Area Network). Colleges, universities, and companies often run computer labs with Windows or Mac stations that can access Linux servers and display their desktop screens.

In any case, enter your correct password carefully. If you are a new user and, after several careful tries, you are unable to log in, it may be that the system administrator has not yet established your userid on the computer. Wait a reasonable length of time and try again. If you still have a problem, contact your system administrator.

After login, you'll see your *desktop* displayed (Figure 1.2). The desktop enables the use of full-GUI (Graphical User Interface) applications that allow point-and-click operations with the mouse.

FIGURE 1.2: A Typical Desktop

Usually, you will be able to choose between two major desktop environments, *GNOME* or *KDE*, to control and manage your *login session*. From the desktop, you can click on the *start icon*, usually a Linux distribution logo located on the left end of your *desktop Panel* (normally a horizontal bar across the top or bottom of your screen) to pull up the `Start` menu listing many tasks you can do.

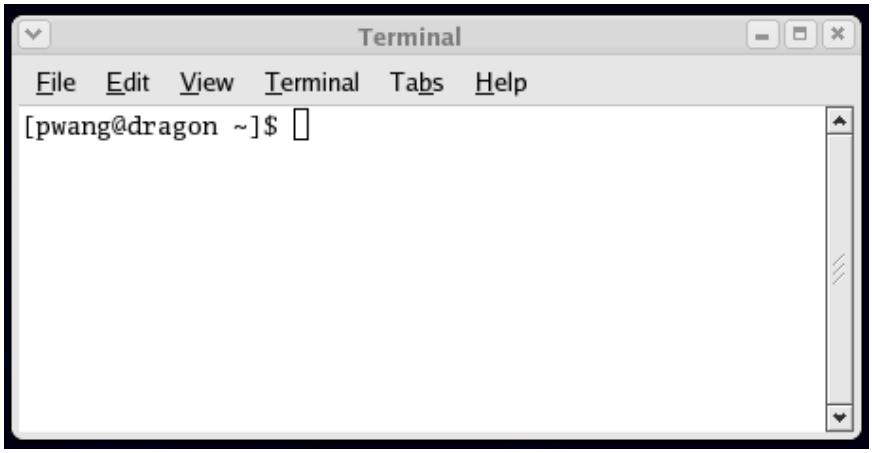
To log out from Linux, you may use the `Start` menu `logout` option. Sometimes, a *logout icon* is present on the desktop Panel to make logout easy.

More will be said about desktops in Chapter 3.

Starting a Terminal Window

From the desktop, you can conveniently initiate many operations including starting a terminal window (Figure 1.3) that runs a Shell (Section 1.3). The Shell provides you with a *command-line interface* (CLI) where you can enter commands to perform almost any task on Linux quickly and efficiently.

FIGURE 1.3: A Terminal Emulation Window



To start a terminal window, go to the `Start` menu and click on the `System tools->Terminal` option or the `Accessories->terminal` option, depending on your Linux and desktop. For the GNOME desktop, the terminal would be **gnome-terminal**, and for KDE it would be **konsole**. A terminal window emulates a character-based computer terminal and allows you to use Linux through a *command interpreter* called the *Shell* (Section 1.3).

As it starts, the Shell also positions you at your *home directory* (Section 1.4), the file folder reserved for you as a user on Linux. The *Shell* indicates its readiness to take your commands by displaying a *prompt* at the beginning of a line.

When you are finished with a terminal window, you may close it by

exit	(exits from Shell and closes the terminal window)
logout	(same as **exit**)

The character CTRL+D (the character `d` typed while holding down the CTRL key) typed alone on a command line often can be used in place of the **exit** command. Exit with CTRL+D is convenient but dangerous, because one typing error can close your terminal window. See Chapter 2 for how to disable exit via CTRL+D.

By the way, we shall use the notation

CTRL+X

to denote a control character, where X is some character. Note also that although the convention is to show an uppercase character, you do not need to hold down SHIFT when typing a control character.

Remember, you need a terminal window to use a Shell, which is your CLI to interact with Linux. Thus, it is important to call up a terminal window quickly and easily. You can create a *keyboard shortcut* to run a terminal window. Go to the `Start` menu

```
Start->System->Preferences->Personal->Keyboard Shortcuts
```

to see all the defined keyboard shortcuts. Find `Run a terminal` and click on that row. The key you type after seeing `New accelerator` will become the keyboard shortcut.

Remote Login

Universities and other institutions often run large Linux servers for users to access through a LAN or even the Internet. You can use TELNET, or more likely SSH (Secure Shell), to access a Linux system from another computer, which can be a PC, another Linux system, or any other platform. Figure 1.4 shows SSH access to a Linux host `monkey.cs.kent.edu` from MS Windows®.

On Linux, the Shell-level command **ssh** provides SSH and is used to access a remote Linux server from a Linux system. For example,

ssh `pwang@monkey.cs.kent.edu`
or
ssh `-X pwang@monkey.cs.kent.edu`

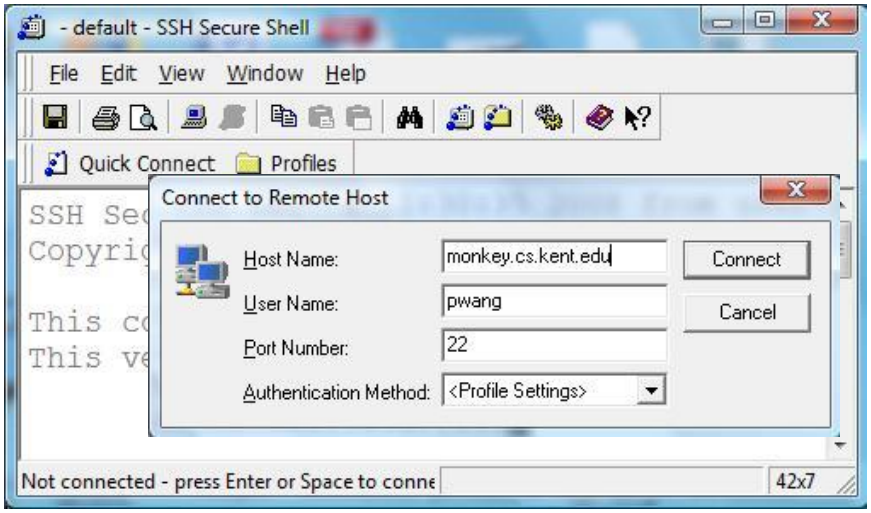
networks to the computer `monkey.cs.kent.edu` (the domain name of the computer) and attempts to log in with the userid `pwang`. Remote login normally supports only CLI access. The `-X` (capital `X`) option allows the remote computer to open the graphical display on the local Linux and therefore enables you to also launch remote applications that require a GUI. Running GUI programs remotely involves much heavier network traffic and can be slow.

Without the `-X` option you'll be able to run only command-line applications on the remote computer which is usually the efficient and sensible thing to do. We will return to SSH in Chapter 7 (Section 7.6) where networking is discussed. Download, installation, and usage information for SSH/SFTP can be found appendices on the companion website (`ml.sofpower.com`).

Successful remote login via SSH results in your SSH window being connected to a *login Shell* running on the remote Linux. After login, Linux will record your login in a system log, display a message showing the time and place for your last login, and initiate a Shell to take your commands.

When you see the prompt, you are ready to begin computing. After you are done, you will need to log out from the remote Linux. To log out, first close any programs that you have been running and then issue the Shell-level command **exit** or **logout**. It is a good practice to first close all running

FIGURE 1.4: Login via SSH



programs manually instead of relying on the logout process to close them for you. It is also good to turn off the power to the display to save energy.

1.3 Understanding the *Shell*

The Shell displays a prompt to signal that it is ready for your next command, which it then interprets and executes. On completion, the Shell resignals readiness by displaying another prompt.

There are several available Shells: the original Shell written by S. R. Bourne known as the *Bourne Shell* or *Sh*, the C-Shell or *Csh* developed at UCB by William Joy, and an enhanced Csh named *Tcsh*. The standard Shell for Linux is the *Bash* (Bourne-Again Sh), which is a powerful and much improved version of Sh. The default Shell on most Linux systems is Bash.

At the Shell prompt, enter the command

echo `$0`

to display the name of the Shell you are using. Here **echo** displays the value of the Shell variable `$0`. Don't worry, Chapter 2 explains how this works.

You can change the default Shell with the **chsh** (change Shell) command. For security reasons, usually only approved Shells can be used. In this text we will assume the Bash Shell, although basic features of all Shells are very similar.

Entering Commands

In Linux, you can give commands to the Shell to start application programs, manage files and folders, control multiple jobs (tasks that are running), redirect I/O of programs from/to files, connect one program to another, and perform many other tasks. Virtually anything you want done in Linux can be accomplished by issuing a command to the Shell.

Many different commands are available, but some general rules apply to all of them. One set of rules relates to *command syntax*—the way the Shell expects to see your commands. A command consists of one or more words separated by blanks. A *blank* consists of one or more spaces and/or tabs. The first word is the *command name* (in this book the name of a command will appear in **boldface**); the remaining words of a command line are *arguments* to the command. A command line is terminated by pressing the RETURN (or ENTER) key. This key generates a NEWLINE character, the actual character that terminates a command line. Multiple commands can be typed on the same line if they are separated by a semicolon (`;`). For example, the command

ls *folder*

lists the names of files in a folder (directory) specified by the argument *folder*. If a directory is not given, **ls** lists the *current working directory* (Section 1.4).

Sometimes one or more *options* is given between the command name and the arguments. For example,

ls `-F` *folder*

adds the `-F` (*file type*) option to **ls** telling **ls** to display the name of each file, or each *filename*, with an extra character at the end to indicate its file type: `/` for a folder, `*` for an executable, and so on.

At the Shell level, the general form for a command looks like

command-name `[` *options* `]` ... `[` *arg* `]` ...

The brackets are used to indicate *optional* parts of a command that can be given or omitted. The ellipses (...) are used to indicate possible repetition. These conventions are followed throughout the text. The brackets or ellipses themselves are not to be entered when you give the command.

Command options are usually given as a single letter after a single hyphen (`-`). For example, the *long listing option* for the **ls** command is `-l`. Such single-letter options can sometimes be hard to remember and recognize. Many Linux commands also offer full-word options given with two hyphens. For example, the `--help` option given after most commands will display a concise description of how to use that particular command. Try

ls `--help`

to see a sample display.

After receiving a command line, the Shell processes the command line as

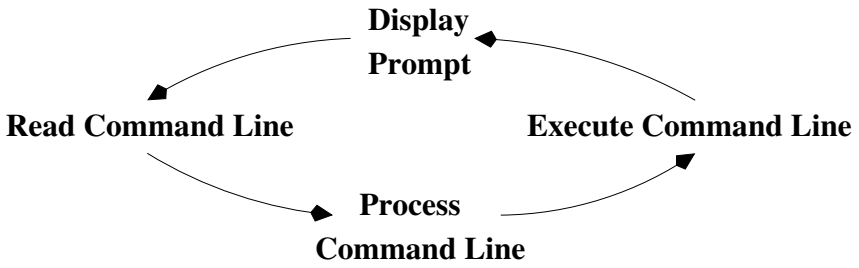
a character string, transforming it in various ways. Then, the transformed command line is executed. After execution is finished, the Shell will display a prompt to let you know that it is ready to receive the next command. Figure 1.5 illustrates the Shell command interpretation loop. *Type ahead* is allowed, which means you can type your next command without waiting for the prompt, and that command will be there when the Shell is ready to receive it.

Trying a Few Commands

When you see the Shell prompt, you are at the Shell level. Now type

```
echo Hello Linux
```

FIGURE 1.5: Command Interpretation Loop



You'll see that the **echo** command displays what you type. Next, enter

```
echo -n "Hello Linux "; echo user
```

This command line contains two commands separated by the `;` command separator. (If you make a mistake typing these commands, glance ahead to the next subheading on correcting typing mistakes.) The option `-n` causes **echo** to omit a NEWLINE character at the end of its output, so the word `user` appears on the same line as `Hello Linux`. Note also the use of quotation marks for the string `Hello Linux` which has a trailing SPACE.

One use of **echo** is to examine the value of a *Shell variable*. For example, if you type

```
echo $HOME
```

you'll see the value of the Shell variable `HOME` which is the location of your home directory in the file system. Note that the *value* of a Shell variable is obtained by prefixing the variable name with a dollar sign (`$`). More on Shell variables can be found in Chapter 2.

A computer on a network is known as a *host* and is usually identified by a *hostname*. To find out your Linux system's hostname, give the command

hostname

To identify the operating system version running on your computer, enter the command

uname `--all`

Another command is **who**. Type

who

to list current users signed in on the system. This gives you an idea of how many people are sharing the computing facility.

The **ls** command will not list *hidden files*, any file whose name begins with the period (`.`) character, unless the `-a` option is given.

ls `-a`

lists the names of all your files, including the hidden ones. Hidden files are usually standard operating system or application files for configuration or other prescribed purposes and ought not be mixed with other files created by the user.

For the Bash Shell, one standard file is `.bash_profile` in a user's home directory. You can place in this file your personal initialization to be used when **bash** starts as a login Shell.

If you are curious about what's in the file `bash_profile.`, type the command

more `.bash_profile`

to display its contents. Press SPACE to continue to the next page or `q` to quit from the **more** display. Don't be discouraged by what you don't understand in this file. When you have progressed further in this book, the contents will become clear.

The Linux system keeps track of the time and date precisely, as you would expect any computer to do. The command

date

displays the current date and time as given by the following typical output showing Eastern Daylight Time

```
Thu Dec  4 16:37:07 EST 2008
```

The Linux system has a dictionary of words for spell checking purposes. The command

spell *file*

will display suspected misspellings for you. Or you can use

aspell `-c` *file*

to interactively spell check the given *file*. To look for words, you can use

look *prefix*

on most Linux systems, and all words in the dictionary with the given prefix are displayed.

Another useful command is **passwd**. Type

passwd

to change your password. This command will prompt as follows

```
Changing password for your userid
Old password:
New password:
Retype new password:
```

pausing after each prompt to wait for input. Many Linux installations give out new userids with a standard password, and the new user is expected to use the **passwd** command to change to a personal password as soon as possible.

The command **man** consults the on-line manual pages for most commands. Thus,

man *command*

will display the on-line documentation for the given command. Try

man `passwd`

just to see what you get. Learn about **man** with

man `man`

Details on the **man** command can be found in Section 1.11.

The **man** command documents *regular commands* (application programs), but normally not commands built in to Shells or other application programs. For Bash you can use

help *builtin_command*

to see a summary of any particular Bash built-in command. Many Linux systems add a `Bash_Builtins` man page so the **man** command will work for Bash built-in commands as well.

Correcting Typing Mistakes

As you entered the preceding commands, you may have made at least one keystroke error, or you may wish to reissue a command you have entered previously. Linux Shells provide easy ways to correct typos and to reuse previous commands. Basically, you can use the arrow keys to move the character cursor left and right on a command line and up to a previous command or down to the next command.

The DELETE (BACKSPACE) key deletes the character under (before) the cursor. The ENTER (RET) key issues the command no matter where the cursor is on the line.

The Bash Shell has great support for editing the command line. It actually allows you to pick a text editor to help do the job. We will return to this in Chapter 2, Section 2.3.

Aborting a Command

Apart from correcting typing mistakes, you can also exercise other controls over your interaction with Linux. For instance, you may abort a command before it is finished, or you may wish to halt, resume, and discard output to the terminal window.

Sometimes, you may issue a command and then realize that you have made a mistake. Perhaps you give a command and nothing happens or it displays lots of unwanted information. These are occasions when you want to abort execution of the command.

To abort, simply type the *interrupt character*, which is usually CTRL+C. This interrupts (terminates) execution and returns you to the Shell level. Try the following

1. Type part of a command.
2. Before you terminate the command, press CTRL+C.

It cancels the command and gives you a new prompt.

Exercise A

1. How do you start a terminal window?
2. What command and option should be used to list all the files in your home directory?
3. Set up `ctrl+alt+T` as the keyboard shortcut for running a terminal window.
4. What command is used to change your password? Can you change you password to something like `123`? Why? Make up a longer password and

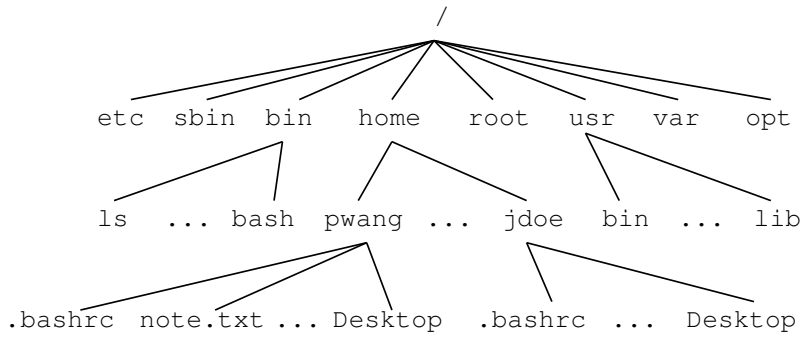
change your password to it. Why did you have to type your password twice this time?

5. Try input editing with the arrow keys under Bash. After doing a command **ls -l**, press UP-ARROW once and LEFT-ARROW twice. Where is the cursor now? Now, press RIGHT-ARROW once and the cursor should be over the letter `l` which is the last character on the command line. Can you press RIGHT-ARROW again to move the cursor beyond `l`? If not, can you find a way? (Hint: Limit yourself to using only the arrow keys.)

6. What is the hostname of your Linux computer? How do you obtain this information?

1.4 Using Files and Directories

FIGURE 1.6: A Sample File Tree



Like other modern operating systems, Linux stores files for users, applications, and the operating system itself on hard disks for ready access. The structure used to store and manage such files is called a *file system*. Files under Linux are organized into a tree structure with a root named by the single character `/`.

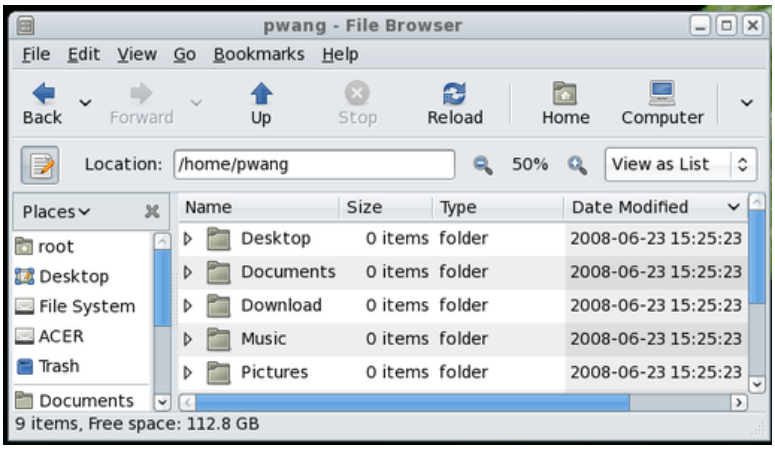
A *regular file* stores a program or data. A *directory* or *folder* contains files and possibly other directories. Internal nodes on the Linux file tree represent directories; leaf nodes represent regular files. This *hierarchical* file structure is widely used by different operating systems. A sample Linux file tree is shown in Figure 1.6.

By clicking on the `computer` icon then the `File System` link, you can launch a visual *file browser* (Figure 1.7) utility which allows you to navigate the file system and perform operations on files and folders. The way to reach the *file browser* may depend on the Linux version you use. While the *file browser*

makes moving about the file system more visual, many Linux users still find dealing with files and folders via the Shell command line more efficient.

Current Working Directory and Filenames

FIGURE 1.7: Linux File Browser



When you get a userid and account on your Linux system, you are given a personal file directory known as your *home directory*. Your home directory will have your userid as its name, and it will usually be a child of a directory called `home`. Your files and folders are kept in your home directory.

To access a file or directory in the file system from the command line, you must call it up by its name, and there are several methods to do this. The most general, and also the most cumbersome, way to specify a *filename* is to list all the nodes in the path from the root to the node of the file or directory you want. This path, which is specified as a character string, is known as the *absolute pathname*, or *full pathname*, of the file. After the initial `/`, all components in a pathname are separated by the character `/`. For example, the file `note.txt` in Figure 1.6 has the absolute pathname

```
/home/pwang/note.txt
```

The full pathname is the complete name of a file. As you can imagine, however, this name often can be lengthy. Fortunately, a filename also can be specified relative to the *current working directory* (also known as the *working directory* or *current directory*). Thus, for the file `/home/pwang/note.txt`, if the current working directory is `/home`, then the name `pwang/note.txt` suffices. A *relative pathname* gives the path on the file tree leading from the working directory to the desired file. The third and simplest way to access a file can be used when the working directory is the same as the directory in which the file is stored. In this case, you simply use the filename. Thus, a Linux file has three names

- A full pathname (for example, `/home/pwang/note.txt`)
- A relative pathname (for example, `pwang/note.txt`)
- A (simple) name (for example, `note.txt`)

The ability to use relative pathnames and simple filenames depends on the ability to change your current working directory. If, for example, your working directory is `/tmp` and you wish to access the file `note.txt`, you may specify the absolute pathname

```
/home/pwang/note.txt
```

or you could change your working directory to `pwang` and simply refer to the file by name, `note.txt`. When you log in, your working directory is automatically set to your home directory. The command

pwd (print working directory)

displays the absolute pathname of your current working directory. The command

cd *directory* (change working directory)

changes your working directory to the specified directory (given by a simple name, an absolute pathname, or a relative pathname).

Two *irregular files* are kept in every directory, and they serve as pointers

File `.` is a pointer to the directory in which this file resides.
File `..` is a pointer to the *parent* directory of the directory in which this file resides.

These pointers provide a standard abbreviation for the current directory and its parent directory, no matter where you are in the file tree. You also can use these pointers as a shorthand when you want to refer to a directory without having to use, or even know, its name. For example, the command

cd .

has no effect, and the command

cd ..

changes to the parent directory of the current directory. For example, if your working directory is `jdoe`, and you want to access the file `sort.c` in the `pwang` directory, you may use `../pwang/sort.c`. Why does this work?

Your home directory already comes with a name, your userid. However, you name your files and subdirectories when you create them. Linux is lenient when it comes to restrictions on filenames. In Linux you may name your file with any string of characters except the character `/`. But, it is advisable to avoid white space characters and any leading hyphen (`-`).

Handling Files and Directories

Generally, there are two kinds of regular files: text and binary. A Linux text file stores characters in ASCII or UNICODE and marks the end of a line with the NEWLINE character.[1] A binary file stores a sequence of bytes. Files may be copied, renamed, moved, and destroyed; similar operations are provided for directories. The command **cp** will copy a file and has the form

cp *source destination*

The file *source* is copied to a file named *destination*. If the destination file does not exist, it will be created; if it already exists, its contents will be overwritten. The **mv** (move) command

mv *oldname newname*

is used to change the file *oldname* to *newname*. No copying of the file content is involved. The new name may be in a different directory—hence the name "move." If *newname* already exists, its original content is lost.

Once a file or subdirectory has outlived its usefulness, you will want to remove it from your files. Linux provides the **rm** command for files and **rmdir** for directories

rm *filenamel filename2 ...*
rmdir *directoryname1 directoryname2 ...*

The argument of **rm** is a list of one or more filenames to be removed. **rmdir** takes as its argument a list of one or more directory names; but note, **rmdir** only will delete an empty directory. Generally, to remove a directory, you must first clean it out using **rm**.

To create a new directory, use the **mkdir** command, which takes as its argument the name of the directory to be created

mkdir *name*

When specifying a file or directory name as an argument for a command, you may use any of the forms outlined. That is, you may use either the full pathname, the relative pathname, or the simple name of a file, whichever you prefer.

Standard Personal Directories

It is easy to change to a home directory, just do

cd	(goes to your home directory)
cd ~*userid*	(goes to the home directory of *userid*)

[1] On Windows or DOS systems, end of line is indicated by RETURN followed by NEWLINE.

In Linux, there are a number of standard folders under each user's home directory, usually including

- `Desktop`—Files in this folder appear as icons on your graphical desktop display, including regular files and *application launchers* (with filename suffix `.desktop`)
- `Documents`—Textual documents such as PDF files and those created using tools such as **openoffice.org**
- `Download`—Files downloaded from the network
- `Music`—Sound and music files
- `Pictures`—Pictures from digital cameras
- `public_html`—Files under this folder are made available to the Web via an HTTP server on your Linux system
- `Videos`—Files from video cameras and recorders

In addition to these, you may consider setting up a `bin/` for your own executables, a `tmp/` for temporary files, a `templates/` for reusable files, a `homework/\` for your classes, and so on.

1.5 Protecting Files: Access Control

Every file has an owner and a group designation. Linux uses a 9-bit code to control access to each file. These bits, called *protection bits*, specify access permission to a file for three classes of users. A user may be a *super user*, the owner of a file, a member in the file's group, or none of the above. There is no restriction on super user access to files.

`u` (The owner or creator of the file)
`g` (Members in the file's group)
`o` (Others)

The first three protection bits pertain to `u` access, the next three pertain to `g` access, and the final three pertain to `o` access. The `g` type of user will be discussed further in Chapter 8.

Each of the three bits specifying access for a user class has a different meaning. Possible access permissions for a file are

`r` (Read permission, first bit set)
`w` (Write permission, second bit set)
`x` (Execute permission, third bit set)

The Super User

Root refers to a class of super users to whom no file access restrictions apply. The *root* status is gained by logging in under the userid `root` (or some other designated root userid) or through the **su** command. A *super user* has read and write permission on all files in the system regardless of the protection bits. In addition, the super user has execute permission on all files for which anybody has execute permission. Typically, only system administrators and a few other selected users ("gurus" as they're sometimes called) have access to the super user password, which, for obvious reasons, is considered top secret.

Examining the Permission Settings

The nine protection bits can be represented by a 3-digit octal number, which is referred to as the *protection mode* of a file. Only the owner of a file or a super user can set or change a file's protection mode; however, anyone can see it. The **ls** `-l` listing of a file displays the file type and access permissions. For example,

```
-rw-rw-rw- 1 smith 127 Jan 20 1:24 primer
-rw-r--r-- 1 smith  58 Jan 24 3:04 update
```

is output from **ls** `-l` for the two files `primer` and `update`. The owner of `primer` is `smith`, followed by the date (January 20) and time (1:24 A.M.) of the last change to the file. The number 127 is the number of characters contained in the file. The *file type*, *access permissions*, and *number of links* precede the file owner's userid (Figure 1.8). The protection setting of the file `primer` gives read and write permission to `u`, `g`, and `o`. The file `update` allows read and write to `u`, but only read to `g` and `o`. Neither file gives execution permissions. There are ten positions in the preceding mode display (of **ls**). The first position specifies the file type; the next three positions specify the `r`, `w`, and `x` permissions of `u`; and so on (Figure 1.8). Try viewing the access permissions for some real files on your system. Issue the command

ls `-l /bin`

to see listings for files in the directory `/bin`.

FIGURE 1.8: File Attributes

file	user	group	other						file
type	access	access	access	links	userid	size	date	time	name
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
-	rw-	r--	r--	1	smith	127	Jan 24	2:04	update

Setting Permissions

A user can specify different kinds of access not just to files, but also to directories. A user needs the `x` permission to enter a directory, the `r` permission to list filenames in the directory, and the `w` permission to create/delete files in the directory.

Usually, a file is created with the default protection

```
-rw-------
```

so only the file owner can read/write the file. To change the protection mode on a file, use the command

chmod *mode filename*

where *mode* can be an octal (base 8) number (for example, 644 for `rw-r--r--`) to set all 9 bits specifically or can specify modifications to the file's existing permissions, in which case *mode* is given in the form

who op permission op2 permission2 ...

Who represents the user class(es) affected by the change; it may be a combination of the letters `u`, `g`, and `o`, or it may be the letter `a` for all three. *Op* (operation) represents the change to be made; it can be `+` to add permission, `-` to take away permission, and `=` to reset permission. *Permission* represents the type(s) of permission being assigned or removed; it can be any combination of the letters `r`, `w`, and `x`. For example,

chmod `o-w` *filename*
chmod `a+x` *filename*
chmod `u-w+x` *filename*
chmod `a=rw` *filename*

The first example denies write permission to others. The second example makes the file executable by all. The third example takes away write and grants execute permission for the owner. The fourth example gives read and write permission (but no execute permission) for all classes of user (regardless of what permissions had been assigned before).

A detailed discussion on the Linux file system can be found in Chapter 6.

Exercise B

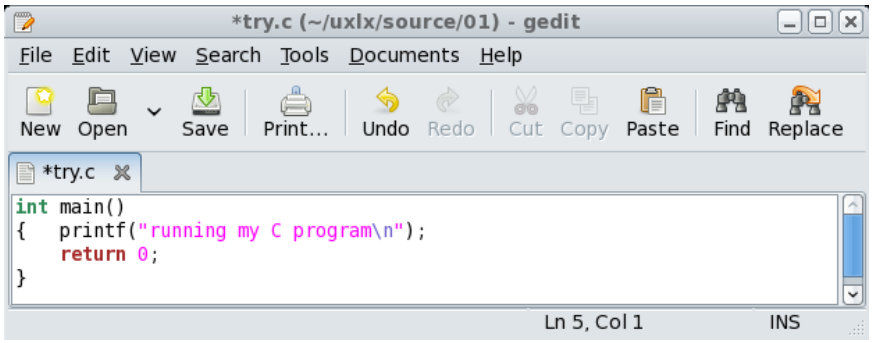
1. Go to your home directory and list all files (hidden ones included) together with the permission settings.
2. Using the **ls** command, list your files in time order (most recent first).
3. List the permission settings of your home directory. Use the **chmod** command to make sure to forbid read and write from `g` and `o`.

4. Create a folder `public_html` directly under you home directory and make sure you open read and execute permissions on this folder.

5. Connect your digital camera to your Linux box and download pictures. Where are the pictures placed. Can you find them under your `Pictures` folder?

1.6 Text Editing

FIGURE 1.9: Gedit



Creating and editing text files is basic to many tasks on the computer. There are many text editors for Linux, including **gedit**, **nano**, **vim**/**gvim**/**vi**, and **emacs**.

The editor **gedit** (Figure 1.9) comes with the GNOME desktop. It requires almost no instructions to use. Start it from the `Start` menu `Text Editor` or the command

gedit *file* `&`

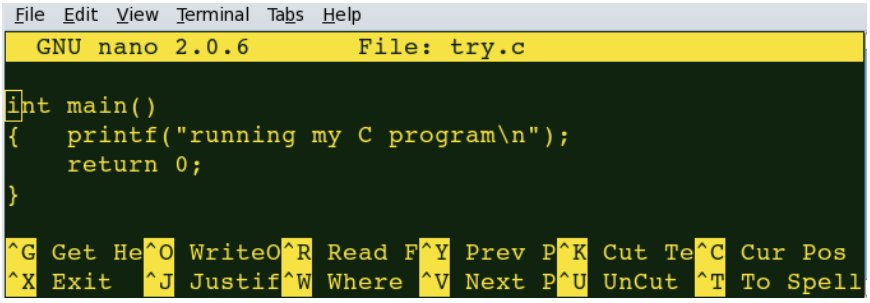
An editor window will display. Then you can type input; move the cursor with the arrow keys or mouse; select text with the mouse; remove text with the DELETE or BACKSPACE key; and find, cut, copy, and paste text with the buttons provided or with the `edit` menu options. It is very intuitive.

The **gedit** is a GUI application. If you want a terminal-window–based editor then consider **nano**, which is very easy to learn but is less powerful or convenient than **vim** or **emacs**. Guides to **vim** and **emacs** can be found in the appendices on the companion website (`ml.sofpower.com`).

Editing power aside, there is something to be said about an editor that is easy and intuitive for simple tasks, especially if you are a beginner. In any case, pick a text editor and learn it well. It can make life on Linux so much easier.

To invoke the editor **nano** for editing *file*, type from the Shell level

FIGURE 1.10: Nano



nano *file* (starts **nano**)
nano `-w` *file* (starts **nano** without line wrapping)

If the file exists, **nano** displays it for editing. Otherwise, you are creating a new file by that name. As you enter text, **nano** will start a new line automatically when the text line gets close to the right edge of your editor window. The `-w` option asks for no such automatic line wrapping. It is also advisable to always use the `-z` option which allows you to suspend **nano** and get back to the Shell level.

Once inside **nano**, you are working in a text-editing environment controlled by **nano**, and you can create text, make changes, move text about, and so on. Common operations are indicated by the **nano** editor window (Figure 1.10). Here is a list to get you started.

- To save the file, type CTRL+O.
- To quite and terminate **nano**, type CTRL+X. You can then elect whether to save the buffer or cancel to change your mind about quiting.
- To move the cursor, use the arrow keys.
- To cut and paste whole lines, CTRL+K cuts one line at a time and CTRL+U pastes the lines cut.
- To cut and paste selected text, type CTRL+6, move the cursor to highlight selected text, and then use CTRL+K and CTRL+U.
- To look for text in the editing buffer, type CTRL+W (where), the text to find, and ENTER or RETURN.
- To get help on operations, type CTRL+G.

1.7 Getting Hard/Saved Copies

To get a printed copy of a file use

lpr [*options*] *filename*

This command sends *filename* to a printer. Your printing request joins a queue of such requests that are processed in order. Note that only text files (plain text, postscript, or pdf) can be printed this way. Do not send a binary file, such as a compiled program (`.o` file), or a compressed file to a printer this way. The `print` option on the `file` menu of application programs, such as your Web browser, PDF (Portable Document Format) reader, or document editor (**openoffice.org** for example), can also be used.

Often, you can avoid wasting paper by using the `print to file` option. You can easily save the resulting file (mostly in PDF) and share with others by email or SFTP (Secure File Transfer Protocol, Chapter 5, Section 5.20).

1.8 Communicating with Others

As soon as you log in, you can potentially interact with others, whether they are users on the same Linux computer or on other *hosts* (computers) connected by networking. Commands such as **who** (who is logged in) and **finger** help to identify members of your user community; email applications allow the sending and receiving of messages and files; and *instant messaging* (IM) programs enable immediate interaction among on-line users anywhere on the Internet.

Who's Who on the System: `finger`

If you are a new user, you may not know many people on the system, and although the information provided by **who** and **w** is useful, you don't know who these users are. You only know their userids, which may not resemble their actual names even faintly. The command **finger** will give you such data as full name, office, address, and phone number for each user; this is sometimes referred to as the *finger database*, because finger is used to look up information from this database. The general form is

finger *name ...*

This command will display all entries in the finger database that contain a userid and first, middle, or last name matching any of the given arguments. For example, either **finger** `smith` or **finger** `clyde` will result in the entry shown in Figure 1.11.

This multiline output includes a *project* field, which is the first line in the `.project` file in the user's home directory. The *plan* field these two files to supply additional information about themselves for the finger database. The

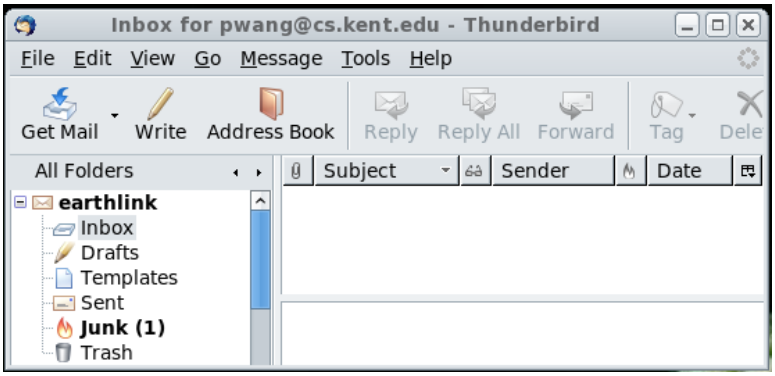
FIGURE 1.11: A Sample **finger** Output

```
Login name: csmith   In real life: Clyde Smith
(803) 555-5432
Directory:/user/grad/csmith   Shell:/bin/bash
Last login Tue May 27 14:49 on ttyhd
Project: Automation Technology Research
No Plan.
```

no plan line in the example indicates that `csmith` has no `.plan` file. On some systems, **finger** gives only a very short summary unless the `-l` option is given.

Used with an argument, **finger** will access information on any user known to the system, whether that user is logged on or not. If **finger** is used without an argument, an abbreviated finger entry is displayed for each user currently logged in. The **f** command is sometimes available as a shorthand for **finger**.

FIGURE 1.12: Thunderbird Email Program



Email

Electronic mail gives you the ability to send and receive messages instantly. A message sent via *email* is delivered immediately and held in a user-specific mailbox for each recipient. You can send email to users on the same computer or on other computers on the Internet.

Many utilities are available on Linux for email, including the popular *Mozilla Thunderbird* (Figure 1.12), *Evolution*, and *Kmail*. These full-GUI email programs are nice when you are at a Linux console. Command-line email programs such as *elm* and *mutt* are useful from a terminal window. Let's explain how to use **mutt**.

mutt *userid*@*host-address* (Internet mail)
mutt *userid* (local mail)

Then just follow instructions and enter the message subject and type/edit your message. **Mutt** lets you edit your message using your favorite text editor. For **mutt** and many other applications that need text editing, set your favorite editor by giving a value to the *environment variable* `EDITOR` (Chapter 2, Section 2.10).

`EDITOR=vim` or `EDITOR=emacs`
export `EDITOR`

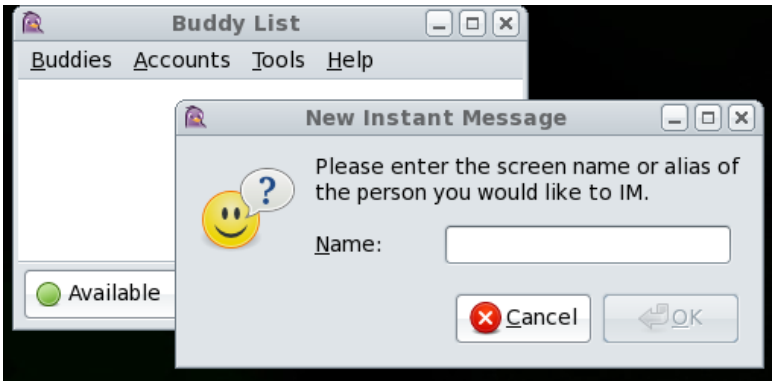
When you finish editing your message, it will be sent out automatically.

mutt `--help | more`

displays more information on **mutt** usage. Here, the output is piped via the `|` notation (Chapter 2, Section 2.5) to the **more** paginator which displays the information one page at a time.

To receive email (to check your mailbox), type **mutt** with no argument and follow instructions. Try to send yourself some email to get familiar with the usage.

FIGURE 1.13: IM with Pidgin



Instant Messaging

Email is fast, but not instant or interactive. On Linux, you can do IM. For example, you can use **pidgin** (Figure 1.13) to IM with friends and business contacts who are online and available. Pidgin is a free IM application capable of working with many different IM services, including AIM by AOL, GTalk by Google, and MSN by Microsoft.

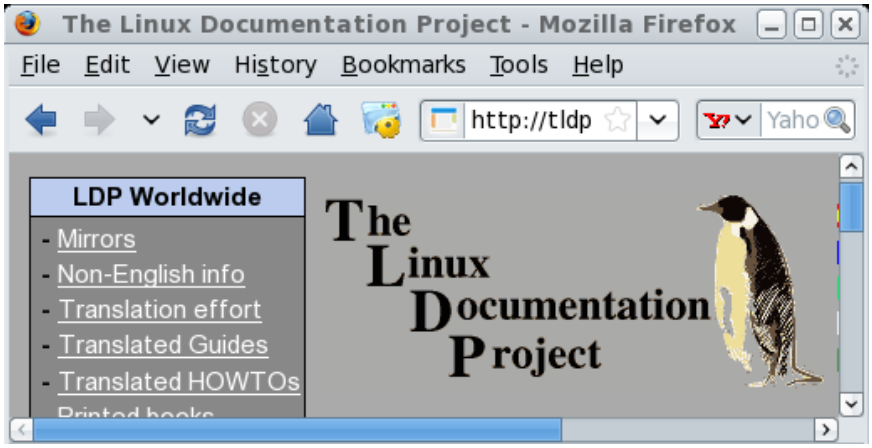
To get started, simply invoke pidgin:

pidgin &

and add your your screen name and password for each IM service you wish to use. Then build up your buddies list and enjoy instant communication whenever you like. The final `&` character causes **pidgin** to run *in the background* so you can use the Shell to perform other tasks.

1.9 Browsing the Web

FIGURE 1.14: Firefox Browser Accessing Linux Documentation



One of the most important tools on any computer system is the Web browser. On Linux you have a choice of different Web browsers. The Mozilla-family browsers include *Mozilla*, *Netscape Navigator*, and *Firefox*. *Chrome* is a fast browser from Google. It is likely that your Linux comes with the Mozilla browser. However, Firefox (Figure 1.14) is a very good browser preferred by many because of its speed, efficiency, and adherence to open Web standards. You can easily download and install Firefox from `mozilla.com`. Then you'll have the command **firefox** to use.

You can enter a URL (*Uniform Resource Locator*) in the browser `Location` window to visit a specific Web address. A local file URL, taking the form `file://`*full_pathname* can be used to visit your local file system.

Normally, Web browsers are full-GUI programs used interactively, but Linux also offers a command-line Web browser called *lynx*, a text-only browser that does not display images. However, lynx can be used inside a terminal window to interactively browse the Web using the arrow keys or to download files from the Web.

Exercise C

1. Try the **mutt** email program. Use it to send an email and attach a file.
2. Create a text file using **nano**.
3. Try the **vi** or **emacs** editor. Read the related appendix on the book's website.
4. If possible, set up Thunderbird as your email program and Firefox or Chrome as your Web browser.
5. Download a file using **lynx** from the Web.

1.10 Creating and Running Your Own Program

Skip this section if you have no immediate interest in writing a program in a general programming language. You can always return to this section later. The Linux system offers many languages: C, C++, Java, Fortran 77/95, Python, Ruby, and Perl, just to name a few. You can also write Shell scripts (Chapter 5) to automate frequently used operations.

Linux follows a set of conventions for naming different kinds of files. Table 1.1 illustrates some commonly used filename suffixes. A source code file cannot be executed directly. The program usually must be compiled into machine code before execution can take place. An alternative to compilation is to interpret a high-level language program directly using an *interpreter*. We shall

TABLE 1.1: File Name Suffixes

Suffix	File Type	Suffix	File Type
`.html`	HTML	`.c`	C source
`.C .cpp`	C++ source	`.java`	Java source
`.f77 .f95`	Fortran source	`.jpg`	JPEG image
`.pdf`	Portable Document Format	`.o`	Object code
`.sh`	Sh or Bash script	`.bash`	Bash script
`.tar`	Tar archive	`.so`	Shared library

follow an example of creating and running a simple C-language program. Use your favorite text editor and create a C source file `try.c` (**Ex:** `ex01/try.c`) with the code

```
#include <stdio.h>

int main()
{   printf("running my C program\n");
    return 0;
}
```

This is a simple source program in C that displays the line "running my first program." The notation `\n` stands for the NEWLINE character.

Compiling

Before `try.c` can be run, it must be compiled. *Compiling* is the process of translating a program written in a high-level language such as C or Pascal into a low-level language for execution on a particular computer. On many systems the compiler will output a file of *object code*, which must be *linked* (combined with routines supplied by the system library) by a separate program called a linker. Once linkage is complete, the file is considered executable and is ready to be loaded into memory and executed.

Linux-based compilers will handle both the compiling and the linking of a program unless you specifically tell them not to, and their output will be an executable file. Available compilers include produce object files (`.o`).

gcc	GNU C compiler
g++	GNU C++ compiler
javac	Java compiler
gfortran	GNU Fortran 77/95 compiler

Let's try compiling the sample program in the file `try.c`

gcc `try.c`

This will produce an executable file `a.out` which can be invoked simply by typing it as a command.

a.out

Note that in Linux the command to run a program is simply the pathname of the executable file. Thus,

./a.out (runs the executable)

At some point, you probably will want to name your executable file something other than `a.out`, especially since `a.out` will be overwritten the next time you invoke a compiler in this working directory. We already know that the **mv** command can be used to rename a file, but the `-o` option to **gcc** or **g++** can be used to provide a name to use instead of the default `a.out`. For example,

gcc `-o mytry try.c`

produces the executable `mytry`.

No matter which language program you run, you probably will want a record of your results (to submit as a homework, for example). One way to do this is to use *output redirection*. For example,

./a.out `> results`

The symbol `>` tells the Shell to *redirect output* of **a.out** from the terminal screen into a new file named `results`. Thus, you will see no output in the terminal window after the preceding command. Instead, you'll find the output in the file `result`. A full account of Shell I/O redirection can be found in Chapter 2, Section 2.5.

Another way to do this is to use the **script** command

script *record_file*

to record your terminal session into a *record_file*. While **script** is active, all I/O to and from your terminal window is written to the file you specified (or to a file named `typescript` if you entered **script** without an argument). Recording stops when you type CTRL+D at the beginning of a command line. The file then can be viewed later with a text editor or emailed to someone. For example, to run a C program with **script**, the following sequence of commands may be used

script `display_record`
cc `myprogram.c`
a.out
CTRL+D

The **script** command requests that all subsequent I/O be recorded in the file `display_record`. The CTRL+D on the last line stops the recording and gets you out of **script** and back to the Shell level.

An advantage of using script over simply redirecting output is that the file produced by script will contain both input to and output from the program. The file created by redirecting output will contain only output.

Exercise D

1. Type in a simple program (in your favorite programming language) to type out the message: `Linux is nice once you know it`. Compile it and run it. Use **script** to get a display record of the program's execution.

2. Use **more** or **nano** to view the file produced by **script** and then send the file to someone by email.

1.11 Consulting Linux Documentation

The *Linux Documentation Project* website `http://tldp.org` (Figure 1.14) provides comprehensive documentation for almost all aspects of Linux. You'll find FAQs, topic-specific step-by-step instructions called *HOWTO*s, guides, and manual pages for commands. A search feature makes it easy to find what you need.

You can also find documentation provided by your own Linux. The `Help` menu on the tool bar of GUI applications, such as the File Browser, the Terminal Emulation Window, and Pidgin, provides tool-specific documentation.

Command-line programs often provide brief and concise usage information with the `--help` command option. For example, try

ls `--help`

The **man** command displays *manual pages* set in standard UNIX-defined formats. Each manual page provides a concise description of a Linux command. The main body of the manual pages is divided into chapters. Although the exact organization of the chapters may vary with the particular Linux system, the following is a typical organization

1. User-level commands
2. Linux system calls in the C language
3. System library functions for C, Fortran, networking, and other purposes
4. Special files, related device driver functions, and networking support
5. Formats for executable and system database files
6. Miscellaneous useful information
7. Linux maintenance, operation, and management

You can use the command

man man

to see the organization of your manual pages. To display an introduction to chapter *n*, type

man *n* `intro`

To display the manual for *command_name*, type

man [*n*] *command_name*

where the chapter number *n* is optional.

When an application or utility is added to your Linux by downloading and installing it, a manual page for that new program is often also added automatically. For example, after installing Firefox, you can do **man** `firefox` to see its manual page.

A typical manual page contains the following information: `NAME` (and principal purpose), usage `SYNOPSIS`, `DESCRIPTION`, `OPTIONS`, related `FILES`, and `SEE ALSO` (related commands).

If the manual page is too large to fit on one screen, the program will display one page at a time until the entire entry has been shown. You can type `q` to quit

man and return to the Shell prompt. This is especially useful if the man page is large and you don't want to see it all. The `SYNOPSIS` part of the manual page gives a concise description of the command syntax. In the synopsis, certain characters are to be typed literally when using the command; other characters or strings are to be replaced by appropriate words or filenames supplied by the user. Portions enclosed in brackets are optional, and the brackets themselves are not part of the command. Ellipses (. . .) are used in the synopsis to indicate possible repetitions. Most Linux commands receive *options* that modify the behavior of the command. As mentioned earlier, an option is usually given as a single character preceded by a dash (`-`), but more verbose options are also possible.

The `FILES` section of the manual page gives the locations of files related to the particular command. The `SEE ALSO` section gives related commands that may be of interest. The `BUGS` section lists some known problems with the command.

The command **man** also can perform a keyword search through the name and purpose part of the manual pages, displaying each line containing any of the given keywords. This is done by using the `-k` option

man `-k` *keyword* . . .

This feature is useful when you want to do something, but can't remember the appropriate command. For example, to figure out how to copy a file, you could try **man** `-k copy`. The *keyword* can be any character sequence. So you can find a command if you remember only a part of its name or description.

There are also Web page versions of the Linux man pages (for example, `linuxmanpages.com`) that can be much easier to use as a reference.

Exercise E

1. How do you ask a command to help you use it?
2. Access the man page for **ls** and read it from beginning to end.
3. Explain how to display the introduction section to the user commands chapter of the Linux man pages.
4. Find and describe a way to do a key-word search of the Linux man pages.
5. Where can you find documentation for a command built in to Bash?
6. Download the most recent Web page version of the Linux man pages from `www.tldp.org/manpages/man-html/` to your computer. It can be much easier to use than the regular man pages. Hint: See Chapter 6, Section 6.12.

1.12 Rounding Up Useful Commands

In this chapter, we have run into only a small number of the hundreds of Linux commands. The richness and variety of Linux commands are major strengths of the system. It is doubtful, however, that many users know all of them; you learn the commands that accomplish what you need. This section collects the commands that should be in a new user's basic repertoire.

In Linux, both uppercase and lowercase characters are used, and they are not interchangeable. All system-defined Linux commands are entered in lowercase. Also, there are two kinds of commands: (1) built-in Shell commands that are subroutines in the Shell and (2) regular commands that are initiated as jobs controlled by the Shell. The importance of this distinction will become clear. In the following listing of commands, user-supplied arguments are shown in *italics*. Optional arguments are enclosed in square brackets [ ]. Possibly repeated arguments are indicated by ellipses (. . .). These conventions will be followed throughout the book. Only the most common usages of these commands are given. The information here is intended to get you started and is by no means complete. Details are provided in later chapters, and you should consult the on-line manual for full descriptions of each of these commands.

File-Related Commands

cat *file* . . .	Displays on the terminal the contents of the file(s) specified in the order they are specified.
more *file*	Displays the file on your terminal, pausing after each complete screenful and displaying `--more--`, at which point you can press SPACE for another screenful, RETURN for another line, or **q** to quit (the command **less** is similar).
cp *filel file2*	Makes a copy of *file1* in *file2*, overwriting it if it exists and creating it otherwise.
cp *file . . . dir*	Makes a copy of the file(s) using the same name in the given directory.
mv *file1 file2*	Renames *file1* as *file2* overwriting it if it already exists.
mv *file . . . dir*	Moves the file(s) to the given directory.
rm *file* . . .	Deletes the files specified.

Login-Related Commands

ssh [*user*`@`]*host*	Remote login.
chsh	Changes your default Shell.
passwd	Changes your password.
exit or **logout**	Terminates the Shell and closes the terminal window.
.	A single dot at the beginning of a line quits the Shell, unless disabled.
su [*user*]	Enters a subshell as *user* or `root`.

Directory-Related Commands

cd *dir*	Changes the current working directory to the given directory (a Shell built-in command).
pwd	Prints the absolute pathname of the current working directory.
mkdir *dir*	Creates a new directory.
rmdir *dir*	Deletes the directory which must be empty.
ls [*op*] [*file* ...]	Lists the filenames in the current working directory if no file argument is given. For each *file* given, if *file* is a directory, then the files in that directory are listed; if *file* is a regular file, then that file is listed. The `-l` option causes the listing to be in "long form." Other options include: `-a`, which lists all files, including those files whose names begin with the period character (`.`); `-d`, which lists the directory entry itself instead of the files in it; and `-F`, which indicates executable files with a trailing asterisk (`*`) and directories with a trailing slash (`/`). Options for **ls** can be combined, as in `-ld`.

Text Editors

gedit *file*	Standard GUI editor from GNU.
nano *file*	Simple and easy terminal-window editor.
vi *file*	Full-featured original UNIX/Linux editor.
vim *file*	Vi improved.
gvim *file*	GUI version of Vim.
emacs *file*	Flexible and powerful editor.

Informational Commands

date	Displays the date and time of day.
finger *name*	Consults the user database for anyone with the userid or *name* given.
look *prefix*	Displays all words from the on-line dictionary having the given prefix.
man *cmd*	Consults the on-line manual pages dealing with the specified command.
who	Displays a list of current users on the system. The command **w** is the same, but gives more information on each user.

Compiling and Running Programs

cc *prog*	Invokes the C compiler on the program (name ending in `.c`), producing an executable file `a.out`.
g++ *prog*	Invokes the C++ compiler on the program (name ending in `.C`), producing an executable file `a.out`.
gfortran *prog*	Compiles the given Fortran 77/95 program.
javac (**java**)	Compiles (runs) Java programs.
a.out	Runs the executable file `a.out`. In general, the file name of an executable file is also the command to execute that file.

Communications and Web Commands

talk *userid*[@*host*]	Initiates a chat.
mutt *userid@host*	Sends email from command line.
pidgin	Sends and receives instant messages.
thunderbird	Sends and receives email.
firefox	Recommended Web browser.
lynx	Terminal-window–based Web browser.

Printing-Related Commands

lpr *file* or (**lp** *file*)	Prints the file on a printer. Linux puts this printing request in a spooling queue. Your file will be printed when it gets to the front of the queue.
lpq (**lpstat**)	Displays the printer queue and job numbers for your print requests.
lprm *job* or **cancel** *job*	Removes your printing request from the queue.
script *file*	Records terminal I/O in the given file. The **script** command creates a subshell to accomplish this task. To stop recording and exit script, use **exit**.

1.13 Summary

Linux provides both full-GUI applications and command-line programs. The GUI is visual and more intuitive to use, but many basic Linux utilities are more convenient on the command line. A Shell (typically *Bash*) running in a terminal window provides a CLI to enter and execute commands. Learning to use both the GUI and the CLI effectively will make life much easier on Linux. The CLI is especially important for remote access of Linux using SSH.

The desktop main menu leads to many useful operations. Chapter 3 presents the Linux desktop. The Shell executes commands you input from the keyboard and displays results in your terminal window. Typing errors can be corrected through input editing.

Both the system and the users store data in files managed by the Linux file system, which has a tree structure. Each file can be referred to by a full (absolute) pathname, a relative pathname, or a simple filename. Each user has a home directory in which personal files and directories can be kept. Files and directories can be created, moved, copied, listed, and destroyed. Read, write, and execute permissions are used to control file access by `u` (owner), `g` (group member), and `o` (others). The owner can set and change the access permissions of a file.

You can communicate directly with other users by using **talk** to chat directly, by email, and by instant messaging (**pidgin**).

Linux offers several text editors. The full-GUI **gedit** is a good choice. For a terminal window, the simple and easy **nano** is good for beginners and light editing tasks. Serious editing is more efficient with an editor such as **vim**. Editing, compiling, and running of a simple C program have been presented.

Linux offers many facilities and a complete set of manuals. The **man** command can be used to consult the manual pages, and the Linux Documentation Project website provides a variety of comprehensive Linux documentations.

Refer to the final section of this chapter for a list of useful commands for Linux beginners.