In this chapter:

- *Working in the Unix Environment*
- *Syntax of Unix Command Lines*
- *Types of Commands*
- *The Unresponsive Terminal*

1

Getting Started

Before you can use Unix, a system staff person has to set up a Unix *account* for you. The account is identified by your *username*, which is usually a single word or an abbreviation. Think of this account as your office—it's your place in the Unix environment. Other users may also be at work on the same system. At many sites, there will be a whole network of Unix computers. So in addition to knowing your username, you may also need to know the *hostname* (name) of the computer that has your account. Alternatively, your account may be shared between all computers on the local network, and you may be able to log into any of them.

Once you've logged in to your account, you'll interact with your system by typing commands at a command line, to a program called a *shell*. You'll get acquainted with the shell, enter a few commands, and see how to handle common problems. To finish your Unix session, you'll log out.

Working in the Unix Environment

Each user communicates with the computer from a terminal. To get into the Unix environment, you first connect to the Unix computer. (Your terminal is probably already connected to a computer.* But Unix systems also let you log into other computers across a network. In this case, log into your local computer first, then use a remote login command to connect to the remote computer. See the section "Remote Logins" in Chapter 6.)

* Some terminals can connect to many computers through a kind of switchboard called a *port contender* or *data switch*. On these terminals, start by telling the port contender which computer you want to connect to.

After connecting your terminal, if needed, you start a session by logging in to your Unix account. To log in, you need your username and a *password*. Logging in does two things: it identifies which user is in a session, and it tells the computer that you're ready to start work. When you've finished, log out—and, if necessary, disconnect from the Unix computer.

^M If someone else has your username and password, they probably can log into your account and do anything you can. They can read private information, corrupt or delete important files, send email messages as if they came from you, and more. If your computer is connected to a network—the Internet or a local network inside your organization—intruders may also be able to log in without sitting at your keyboard! See the section "Remote Logins" in Chapter 6 for one explanation of one way this can be done.

Anyone may be able to get your username—it's usually part of your email address, for instance. Your password is what keeps others from logging in as you. Don't leave your password anywhere around your computer. Don't give your password to anyone who asks you for it unless you're sure they'll preserve your account security. Also don't send your password by email; it can be stored, unprotected, on other systems and on backup tapes, where other people may find it and then break into your account.

If you suspect that someone is using your account, ask system staff for advice. If you can't do that, setting a new password may help; see the section "Changing Your Password" in Chapter 3.

Unix systems are case sensitive. Most usernames, commands, and filenames use lowercase letters (though good passwords use a mixture of lower- and uppercase letters). Before you log in, be sure your CAPS LOCK key is off.

Connecting to the Unix Computer

If you see a message from the computer that looks something like this:

```
login:
```

you're probably connected! You can skip ahead to the section "Logging in Nongraphically" and log in.

Otherwise, if someone nearby uses the same kind of computer system you do, the easiest way to find out if you're connected is probably to ask for help. (We can't cover every user's situation exactly. There are just too many possibilities.)

If there's no one to ask, look ahead at the section "Logging in Nongraphically," later in this chapter, as well as the section "Starting X" in Chapter 2 and the section "Remote Logins" in Chapter 6. You may recognize your situation.

If that doesn't help, but your computer seems to be running an operating system other than Unix (such as Microsoft Windows), check your menus and icons for one with the name of the Unix computer you're supposed to connect to. You might also find a program named either **telnet**, **eXceed**, **ssh**, **VMware**, **procomm**, **qmodem**, **kermit**, or **minicom**, or something relating to remote access.

Logging in Nongraphically

The process of making yourself known to the computer system and getting to your Unix account is called *logging in*. If you've connected to the Unix host from another operating system, you may have been logged into Unix automatically; in this case, you should be able to run Unix programs, as shown later in this chapter in the section "Shells in a Window System" and the section "The Shell Prompt." Otherwise, before you can start work, you must connect your terminal or terminal window to the computer you need (as in the previous section) and identify yourself to the Unix system.

There are generally two ways to log in: graphically and nongraphically. If your screen has a window or windows floating in it, something like Figure 2-2A, you probably need to log in graphically, as explained by "the section "A. Ready to Run X (with a Graphical Login)" in Chapter 2.

Otherwise, to log in nongraphically, enter your username (usually your name or initials) and your private password. The password does not appear as you enter it.

When you have logged in successfully, you'll get some system messages and finally the shell prompt (where you can enter Unix commands). A successful login to the system named *nutshell* could look like Example 1-1.

Example 1-1. Nongraphical login

```
nutshell login: john
Password:
Last login: Mon Oct  8 14:34:51 EST 2001 from joe_pc
Sun Microsystems Inc.   SunOS 5.7       Generic October 1998

------------- NOTICE TO ALL USERS -----------------
The hosts nutshell, mongo, and cruncher will be down
for maintenance from 6 to 9 PM tonight.
-------------------------------------------------

My opinions may have changed, but not the fact that I am right.
Tue Oct  9 12:24:48 MST 2001
$
```

In this example, the system messages include a maintenance notice, a "fortune," and the date. Although this example doesn't show it, you may be asked for your *terminal type*, accounting or chargeback information, and so on. The last line to appear is the Unix shell prompt. When you reach this point, you're logged in to your account and can use Unix commands.

Instead of a shell prompt, you may get a menu of choices ("email," "news," and so on). If one choice is something like "shell prompt" or "command prompt," select it. Then you'll be able to follow descriptions and examples in this book.

The messages you see at login time differ from system to system and day to day. Shell prompts can also differ. Examples in this book use the currency sign `$` as a prompt.

Let's summarize logging in nongraphically, step by step:

1. If needed, connect your terminal or terminal window to the Unix system.
2. Get a "login:" prompt.
3. Type in your username in *lowercase letters* at the prompt. For example, if your login name is "john," type:

   ```
   login: john
   ```

   Press the RETURN key.

   The system should prompt you to enter your password. If passwords aren't used on your system, you can skip the next step.

4. If you were assigned a password, type it at the prompt. For security, your password is not displayed as you type it:

   ```
   Password:
   ```

   Press the RETURN key.

The system checks your account name and password, and if they're correct, logs you in to your account.

Problem checklist

Nothing seemed to happen after I logged in.

Wait a minute, since the system may just be slow. If you still get nothing, ask other users if they have the same problem.

The system says "login incorrect."

If you have a choice of computer to log into (as we explained at the start of this chapter in the section "Working in the Unix Environment"), check that you're connected to the right computer. If you have accounts on several computers, be sure you're using the correct username and password for this computer. Otherwise, try logging in again, taking care to enter the username and password correctly. Be sure to type your username at the "login:" prompt and your password at the "password:" prompt. Backspacing may not work while entering either of these; if you make a mistake, use RETURN to get a new "login:" prompt and try again. Also make sure to use the exact combination of upper- and lowercase letters your password contains.

If you still fail after trying to log in a few more times, check with the person who created your account to confirm your username and password.

All letters are in UPPERCASE and/or have backslashes (\) before them.

You probably entered your username in uppercase letters. Type **exit** at the shell prompt and log in again.

The Unix Shell

Once you have a shell prompt, you're working with a program called a *shell*. The shell interprets command lines you enter, runs programs you ask for, and generally coordinates what happens between you and the Unix operating system. Common shells include Bourne (**sh**), Korn (**ksh**), and C (**csh**) shells, as well as **bash** and **tcsh**.

For a beginner, differences between shells are slight. If you plan to work a lot with Unix, though, you should learn more about your shell and its special commands.*

Shells in a Window System

If you're using a window system, as described in Chapter 2, get a shell by opening a *terminal window*—if you don't already have a terminal window open or iconified (minimized) somewhere, that is. (Figure 2-1 shows an example, but yours may look different; the important thing is that the window have a shell prompt in it.) Check your menus and icons for a command with "terminal" or "term" in its name, or a picture of a blank terminal (like a TV screen) in its icon; one common program is **xterm**.

The Shell Prompt

When the system is ready to run a command, the shell outputs a *prompt* to tell you that you can enter a command line.

Shell prompts usually end with `$` or `%`. The prompt can be customized, though, so your own shell prompt may be different.

A prompt that ends with a hash mark (`#`) usually means that you're logged in as the *superuser*. The superuser doesn't have the protections for standard users that are built into the Unix system. In this case, we recommend that you stop work until you've found out how to access your personal Unix account.†

Entering a Command Line

Entering a command line at the shell prompt tells the computer what to do. Each command line includes the name of a Unix program. When you press RETURN, the shell interprets your command line and executes the program.

---

* To find out which shell you're using, run the commands **echo $SHELL** and **ps $$**. (See the section "Entering a Command Line," later in this chapter.) The answer, something like *bash* or */bin/bash*, is your shell's name or pathname.

† This can happen if you're using a window system that was started by the superuser when the system was rebooted. Or maybe your prompt has been customized to end with `#` when you aren't the superuser.

The first word that you type at a shell prompt is always a Unix command (or program name). Like most things in Unix, program names are case sensitive; if the program name is lowercase (and most are), you must type it in lowercase. Some simple command lines have just one word, which is the program name. For more information, see the section "Syntax of Unix Command Lines," later in this chapter.

date

An example single-word command is **date**. Entering the command **date** displays the current date and time:

```
$ date
Tue Oct  9 13:39:24 MST 2001
$
```

As you type a command line, the system simply collects your keyboard input. Pressing the RETURN key tells the shell that you've finished entering text and that it can run the program.

who

Another simple command is **who**. It displays a list of each logged-on user's username, terminal number, and login time. Try it now, if you'd like.

The **who** program can also tell you who is logged in at your terminal. The command line is **who am i**. This command line consists of the command (**who**, the program's name) and arguments (**am i**). (Arguments are explained in the section "Syntax of Unix Command Lines," later in this chapter.)

```
$ who am i
cactus!john     tty23   Oct  6 08:26      (rose)
```

The response shown in this example says that:

- "I am" John (actually, my username is *john*).
- I'm logged on to the computer named "cactus."
- I'm using terminal 23.
- I logged in at 8:26 on the morning of October 6.
- I started my login from another computer named "rose."

Not all versions of **who am i** give the same information.

Recalling Previous Commands

Modern Unix shells remember command lines you've typed before. They can even remember commands from previous login sessions. This handy feature can save you a lot of retyping common commands. As with many things in Unix, though, there are several different ways to do this; we don't have room to show and explain them all. You can get more information from sources listed in the section "Documentation" in Chapter 8.

After you've typed and executed several command lines, try pressing the up-arrow key on your keyboard. If your shell is configured to understand this, you should see the previous command line after your shell prompt, just as you typed it before. Pressing the up-arrow again recalls the previous command line, and so on. Also, as you'd expect, the down-arrow key will recall more recent command lines.

To execute one of these remembered commands, just press the RETURN key. (Your cursor doesn't have to be at the end of the command line.)

Once you've recalled a command line, you can also edit it. If you don't want to execute any remembered commands, cancel the command line with CTRL-C. Next, the section "Correcting a Command Line" explains both of these.

Correcting a Command Line

What if you make a mistake in a command line? Suppose you typed **dare** instead of **date** and pressed the RETURN key before you realized your mistake. The shell will give you an error message:

```
$ dare
dare: command not found
$
```

Don't be too concerned about getting error messages. Sometimes you'll get an error even if it appears that you typed the command correctly. This can be caused by typing control characters that are invisible on the screen. Once the prompt returns, reenter your command.

As we said earlier (in the section "Recalling Previous Commands") most modern shells let you recall previous commands and edit command lines. If you'll do a lot of work at the shell prompt, it's worth learning these handy techniques. They take more time to learn than we can spend here,

though—except to mention that, on those shells, the left-arrow and right-arrow keys may move your cursor along the command line to the point where you want to make a change. Here, let's concentrate on simple methods that work with all shells.

If you see a mistake before you press RETURN, you can use the *erase character* to erase and correct the mistake.

The erase character differs from system to system and from account to account, and can be customized. The most common erase characters are:

- BACKSPACE
- DELETE, DEL, or RUBOUT
- CTRL-H

CTRL-H is called a *control character*. To type a control character (for example, CTRL-H), hold down the CTRL key, then press the letter "h." In the text, we will write control characters as CTRL-H, but in the examples, we will use the standard notation: `^H`. This is *not* the same as pressing the ^ (caret) key, letting go, and then typing an H!

The key labeled DEL may be used as the *interrupt character* instead of the erase character. (It's labeled DELETE or RUBOUT on some terminals.) This key is used to interrupt or cancel a command, and can be used in many (but not all) cases when you want to quit what you're doing. Another character often programmed to do the same thing is CTRL-C.

Other common control characters are:

CTRL-U
: Erases the whole input line; you can start over.

CTRL-S
: Pauses output from a program that's writing to the screen. This can be confusing; we don't recommend using CTRL-S, but want you to be aware of it.

CTRL-Q
: Restarts output after a pause by CTRL-S.

CTRL-D
: Used to signal end-of-input for some programs (such as **cat** and **mail**, explained in Chapter 1 and Chapter 6) and return you to a shell prompt. If you type CTRL-D at a shell prompt, it may close your terminal window or log you out of the Unix system.

Find the erase and interrupt characters for your account and write them here:

_____ Backspace and erase a character

_____ Interrupt a program

Logging Out

To end a Unix session, you must log out. You should *not* end a session by just turning off your terminal!

If you're using a window system, first close open windows and then close the window system; see the section "Quitting" in Chapter 2 for more information. If you logged in graphically, that should end your login session. But, if you logged in nongraphically before you started the window system, closing the window system should take you back to a shell prompt (where you originally typed **xinit** or **startx**). In that case, use the following steps to finish logging out.

If you aren't currently using a window system, you can log out by entering the command **exit** at a shell prompt. (In many cases, the command **logout** will also work.) Depending on your shell, you may also be able to log out simply by typing CTRL-D.

What happens next depends on the place from which you've logged in: if your terminal is connected directly to the computer, the "login:" prompt should appear on the screen. Otherwise, if you were connected to a remote computer, the shell prompt from your local computer should reappear on your screen. (That is, you're still logged in to your local computer.) Repeat the process if you want to log out from the local computer.

After you've logged out, you can turn off your terminal or leave it on for the next user. But, if the power switch for your terminal is the same as the power switch for the whole Unix computer system, do *not* simply turn off that power switch! Ask a local expert for help with shutting down your Unix system safely.

Problem checklist

The first few times you use Unix, you aren't likely to have the following problems. But you may encounter these problems later, as you do more advanced work.

You get another shell prompt or the shell says "logout: not login shell"
You've been using a subshell (a shell created by your original login shell). To end each subshell, type **exit** (or just type CTRL-D) until you're logged out.

The shell says "There are stopped jobs" or "There are running jobs."
Many Unix systems have a feature called *job control* that lets you suspend a program temporarily while it's running or keep it running separately in the "background." One or more programs you ran during your session has not ended, but is stopped (paused) or in the background. Enter **fg** to bring each stopped job into the foreground, then quit the program normally. (See Chapter 7 for more information.)

Syntax of Unix Command Lines

Unix command lines can be simple, one-word entries such as the **date** command. They can also be more complex; you may need to type more than the command or program name.*

A Unix command may or may not have *arguments*. An argument can be an option or a filename. The general format for Unix command lines is:

command *option(s) filename(s)*

There isn't a single set of rules for writing Unix commands and arguments, but you can use these general rules in most cases:

- Enter commands in lowercase.
- *Options* modify the way in which a command works. Options are often single letters prefixed with a dash (-, also called "hyphen" or "minus") and set off by any number of spaces or tabs. Multiple options in one command line can be set off individually (such as **-a -b**). In some cases, you can combine them after a single dash (such as **-ab**)—but most commands' documentation doesn't tell you whether this will work; you'll have to try it.

  Some commands, including those on Linux systems, also have options made from complete words or phrases and starting with two dashes, like **--delete** or **--confirm-delete**. When you enter a command line, you can use this option style, the single-letter options (which all start with a single dash), or both.

* The command can be the name of a Unix program (such as **date**), or it can be a command that's built into the shell (such as **exit**). You probably don't need to worry about this! You can read more precise definitions of these terms and others in Glossary.

- The argument *filename* is the name of a file that you want to use. Most Unix programs also accept multiple filenames, separated by spaces. If you don't enter a filename correctly, you may get a response such as "*filename*: no such file or directory" or "*filename*: cannot open."

  Some commands, such as **telnet** and **who** (shown earlier in this chapter), have arguments that aren't filenames.

- You must type spaces between commands, options, and filenames.

- Options come before filenames.

  In a few cases, an option has another argument associated with it; type this special argument just after its option. Most options don't work this way, but you should know about them. The **sort** command is an example of this: you can tell **sort** to write the sorted text to a filename given after its **-o** option. In the following example, **sort** reads the file *sortme* (given as an argument), and writes to the file *sorted* (given after the **-o** option):

  ```
  $ sort -o sorted -n sortme
  ```

  We also used the **-n** option in that example. But **-n** is a more standard option; it has nothing to do with the final argument *sortme* on that command line. So, we also could have written the command line this way:

  ```
  $ sort -n -o sorted sortme
  ```

  Another example is the **mail -s** option, shown in the section "Sending Mail from a Shell Prompt" of Chapter 6. Don't be too concerned about these special cases, though. If a command needs an option like this, its documentation will say so.

- Command lines can have other special characters, some of which we see later in this book. They also can have several separate commands. For instance, you can write two or more commands on the same command line, each separated by a semicolon (**;**). Commands entered this way are executed one after another by the shell.

Unix has a lot of commands! Don't try to memorize all of them. In fact, you'll probably need to know just a few commands and their options. As time goes on, you'll learn these commands and the best way to use them for your job. We cover some useful Unix commands in later chapters. This book's quick reference card has quick reminders.

Let's look at a sample Unix command. The **ls** program displays a list of files. You can use it with or without options and arguments. If you enter:

```
$ ls
```

you'll see a list of filenames. But if you enter:

```
$ ls -l
```

there'll be an entire line of information for each file. The **-l** option (a dash and a lowercase letter "L") changes the normal **ls** output to a long format. You can also get information about a particular file by using its name as the second argument. For example, to find out about a file called *chap1*, enter:

```
$ ls -l chap1
```

Many Unix commands have more than one option. For instance, **ls** has the **-a** (*all*) option for listing hidden files. You can use multiple options in either of these ways:

```
$ ls -a -l
$ ls -al
```

You must type one space between the command name and the dash that introduces the options. If you enter **ls-al**, the shell will say "ls–al: command not found."

Exercise: entering a few commands

The best way to get used to Unix is to enter some commands. To run a command, type the command and then press the RETURN key. Remember that almost all Unix commands are typed in lowercase.

Get today's date.	Enter **date**
List logged-in users.	Enter **who**
Obtain more information about users.	Enter **who -u** or **finger** or **w**
Find out who is at your terminal.	Enter **who am i**
Enter two commands in the same line.	Enter **who am i;date**
Mistype a command.	Enter **woh**

In this session, you've tried several simple commands and seen the results on the screen.

Types of Commands

When you use a program, you'll want to know how to control it. How can you tell it what job you want done? Do you give instructions before the program starts, or after it's started? There are three general ways to give commands on a Unix system, three different kinds of programs. It's good to be aware of them.

1. Some Unix programs work only with a window system. For instance, when you type **netscape** at a shell prompt (or click a button or choose the command from a menu), the Netscape web browser starts. It opens one or more windows on your screen. The program has its own way to receive your commands—through menus and buttons on its windows, for instance.

2. You've also seen (previously, in the section "Syntax of Unix Command Lines") Unix commands that you enter at a shell prompt. These programs work in a window system (from a terminal window) or from any terminal. Control those programs from the Unix command line—that is, by typing options and arguments from a shell prompt before you start the program running. After you start the program, wait for it to finish; you generally don't interact with it.

3. Some Unix programs that work in terminals have commands of their own. (If you'd like some examples, see the section "Looking Inside Files with less" in Chapter 3 and the section "The Pico Text Editor" in Chapter 4.) These programs may accept options and arguments on their command line. But, once you start the program, it prints its own prompt and/or menus and it understands its own commands; it takes instructions from your keyboard, which weren't given on its command line.

   For instance, if you enter **pine** at a shell prompt, you'll see a new prompt from the **pine** program. Enter Pine commands to handle email messages. When you enter the special command **q** to quit the **pine** program, **pine** will stop prompting you. Then you'll get another shell prompt, where you can enter other Unix commands.

The Unresponsive Terminal

During your Unix session (while you're logged in), your terminal may not respond when you type a command, or the display on your screen may stop at an unusual place. That's called a "hung" or "frozen" terminal or session.

(Note that most of the techniques in this section apply to terminal windows in a window system, but not to nonterminal windows such as a web browser. In Chapter 2, the section "Unresponsive Windows" should help with windows in general.)

A session can hang for several reasons. For instance, the connection between your terminal and the computer can get too busy; your terminal has to wait its turn. (Other users or computers probably share the same connection.) In that case, your session starts by itself in a few moments. You should *not* try to "un-hang" the session by entering extra commands because those commands will all take effect after the connection resumes.

If the system doesn't respond for quite a while (how long that is depends on your individual situation; ask other users about their experiences), the following solutions usually work. Try the following steps in the order shown until the system responds:

1. Press the RETURN key *once*.

   You may have typed text at a prompt (for example, a command line at a shell prompt) but haven't yet pressed RETURN to say that you're done typing and your text should be interpreted.

2. If you can type commands, but nothing happens when you press RETURN, try pressing LINEFEED or typing CTRL-J. If this works, your terminal needs resetting to fix the RETURN key. Some systems have a **reset** command that you can run by typing CTRL-J **reset** CTRL-J. If this doesn't work, you may need to log out and log back in or turn your terminal off and on again. (But, before you turn off your terminal, read the notes earlier and later in this chapter about turning off the power.)

3. If your shell has job control (see Chapter 7), type CTRL-Z.

   This suspends a program that may be running and gives you another shell prompt. Now you can enter the **jobs** command to find the program's name, then restart the program with **fg** or terminate it with **kill**.

4. Use your interrupt key (found earlier in this chapter in the section "Correcting a Command Line"—typically DELETE or CTRL-C).

   This interrupts a program that may be running. (Unless a program is run in the background, as described in the section "Running a Command in the Background" in Chapter 7, the shell waits for it to finish before giving a new prompt. A long-running program may thus appear to hang the terminal.) If this doesn't work the first time, try it once more; doing it more than twice usually won't help.

5. Type CTRL-Q .

   If output has been stopped with CTRL-S , this will restart it. (Note that some systems will automatically issue CTRL-S if they need to pause output; this character may not have been typed from the keyboard.)

6. Check that the NO SCROLL key (if you have one) is not locked or toggled on.

   This key stops the screen display from scrolling upward. If your keyboard has a NO SCROLL key that can be toggled on and off by pressing it over and over, keep track of how many times you've pressed it as you try to free yourself. If it doesn't seem to help, be sure you've pressed it an even number of times; this leaves the key in the same state it was when you started.

7. Check the physical connection from the terminal to the system.

8. Type CTRL-D *once* at the beginning of a new line.

   Some programs (such as **mail**) expect text from the user. A program may be waiting for an end-of-input character from you to tell it that you've finished entering text. Typing CTRL-D may cause you to log out, so you should try this only as a last resort.

9. If you're using a window system, close (terminate) the window you're using and open a new one. See the section "Unresponsive Windows" in Chapter 2.

   Otherwise, turn your terminal off, wait ten seconds or so, then turn it on again. This may also log you out, but it may not; your old login session could still be running. You can check for old processes and terminate them (as explained in Chapter 7 in the section "Checking on a Process" and in the section "Cancelling a Process")—although this isn't an easy thing for a beginner to do, so you might want help.