

Chapter 1

ARE YOU ONLINE YET?

The first stage of your journey is a mental one. You must accept that Unix is a large, complex system to tackle. Unless you take one careful, diligent step at a time, it is easy to become somewhat overawed. If you flip aimlessly through the mountains of official documentation, you can be forgiven for feeling daunted by the number of commands and options, and by the sheer weight of quirky detail. The good news is that you need learn only a few Unix basics in order to become productive. As you build on this solid foundation, layer by layer, the climb becomes more gentle. This book stresses constant practice with useful examples so that you see cause and effect rather than ponder textbook abstractions. It also emphasizes motivation: knowing *why* the Unix designers provided certain features often clarifies *how* they are implemented. It proves much easier to assimilate facts when they are interrelated in some way, rather than picked up in isolation.

We can certainly discount any rumors that Unix is impossibly arcane unless you have an advanced degree in computer science. The proof can be found in thousands of ordinary office installations worldwide. Although Unix dominates the academic and research arenas, its presence in the business world has been growing rapidly on every platform from mainframe to PC.

GAINING ACCESS—WHAT YOU NEED TO GET STARTED

As with any computer system, there are several prerequisites before you can gain access and start using Unix. You obviously need a suitable terminal correctly connected to a computer that is running Unix. (The latter caution is not as dumb as it seems—many computers nowadays are capable of running several operating systems, so you need to be sure that you are talking to the right one.) Each Unix vendor supplies precise documentation for installing a particular combination of hardware and version of Unix.

When you are connected and ready to go, you are said to be *online* to the computer. When you lose connection for any reason, you go *offline*, and life becomes simple again. Your terminal may be local, directly cabled (or *hard-wired*) to the computer, or it may be remotely connected via modems and telephone lines. The difference affects only the way you establish a connection; once you are online, the type of connection, local or remote, does not normally affect the way you work.

Remote connection is made by dialing the number of the Unix system just as you would make a normal telephone call. In most cases, the dialing is automated with suitable telecommunications software. The actual procedures are very much site-dependent, so you will need to consult your local guru or Unix vendor. Operating remotely, of course, exposes you to the random gremlins of the telecommunications underworld—glitches on the lines, electrical interference, broken connections, and so on—as well as the psychological stresses of isolation if things go wrong.

Terminals

Unix supports almost every type of terminal ever invented, and indeed is designed so that terminals not yet invented can be handled by simple software changes. As far as the beginner is concerned, a terminal can be looked on as a device with a *keyboard* for typing in instructions and data to the

computer (*input*), and a CRT or monitor screen to *echo* your input and display messages or responses from the computer (*output*). The computer itself usually has a hard-wired terminal referred to as the *console* or *systems console*.

Terminals can be *dumb* or *intelligent*, although it is not always easy to tell the difference. A dumb terminal has a keyboard and CRT but no local processing power—it relies on the central computer for all its processing needs. An intelligent (or *smart*) terminal, in addition to a keyboard and CRT, has some built-in processing power and local memory. With the low costs of microprocessors and integrated circuits, the completely dumb terminal is now quite rare. The same cost factor has also led to the widespread use of PCs (personal computers) as very intelligent devices capable of working both online as Unix terminals and offline as stand-alone PCs. Indeed, with the advent of superfast microprocessors such as the Intel Pentium and Motorola G3, which can handle high-capacity disks and lots of RAM, your Unix may well be running on a PC or workstation while supporting other terminals and PCs.

Keyboards

Whether you have a dumb or intelligent terminal, an online PC, or a PC running Unix, you will have a keyboard for input and a screen for output. With all the varieties of keyboard layout and key labeling currently on the market, you will appreciate that the instructions in this book may need some adjustment for your particular keyboard. To further complicate matters, most smart terminals allow you to personalize the keyboard in various ways. In other words, you may be able to *remap* the keyboard, thereby changing the action of certain keys or sequences of keys.

An extreme form of remapping is the use of *emulation* software that makes a brand X terminal behave like a brand Y model. Whatever tricks you perform, Unix needs to know the type and mode of the terminal being used so it can correctly interpret the stream of characters you are sending. The normal alphanumeric printable characters are fairly standard (usually the so-called ASCII set), but the invisible control characters, like Ctrl, Alt, Shift and Insert, are subject to the more diverse interpretations of the hardware jungle of keyboards, screens, and (especially) printers. The parts of Unix that cope with these vagaries are, perforce, correspondingly messy. Until you know more, therefore, it is best to have your vendor or system administrator set up Unix for your particular terminal.

To avoid too many diversions, I will assume the standard PC keyboard conventions. When we encounter the need for special keystrokes other than the universal alphanumeric keys, I will point out some of the most common variants. For example, the key called Enter on the PC plays an essential role in Unix input. Your Enter key may be marked as Return, CR, Car Ret, or ↵.

Screens

Although Unix was originally developed for systems using monochrome, text-based displays, today most versions of Unix use some kind of GUI (Graphical User Interface). GUIs are, like Microsoft Windows, designed to make using the computer more intuitive to you, the user.

This book was written for monochrome text displays using standard 25-line- \times -80-column displays. Unless specifically stated otherwise, this is the configuration you should be using to work through the examples in this book. The reason we use a text-based approach instead of a GUI-based approach is twofold. First, virtually every one of the functions and commands for operating Unix are text-based or have a text-based equivalent. And second, although Unix has many graphical operations that correspond to the text commands, there are many that have no graphical equivalent and would therefore be impossible to show in a graphics-only environment. Additionally, although almost all Unix systems today use some variation of X-Windows for their GUI, there are several variations that could make these initial descriptions quite difficult to follow. As such, we'll leave the discussion of how X-Windows operates for later on in this book. For the moment, you need to get to a text window to proceed.

To access a text window, you should do one of the following:

- ▶ Open a Terminal window on your GUI. In most X-Window systems, when you initially log in, the system will automatically open a terminal window for you in the GUI. If your system does not automatically open a terminal window for you, you can open one by clicking the appropriate icon on the Control Panel.
- ▶ Switch to a text-only screen. In SCO Unix you can accomplish this by holding down the Ctrl and Alt keys and then pressing one of the function keys: F1–F12.

When you get to the new text screen, you will need to log in again. Don't let this throw you, just enter your login name and password again.

WHO DOES WHAT?

In larger installations, there are staff specially assigned to manage the system, so you would not normally be concerned with such mundane tasks as installing terminals and ensuring that Unix is set to handle them. In particular, you'll find a system administrator (SA for short) who will be the chief liaison between you as user and the Unix system. Whether system administration is a single full-time job or spread among staff members according to shifts or particular responsibilities will depend on the size and complexity of your Unix site. At smaller installations, the system administrator may simply be a designated user selected (or sentenced) to play this role in addition to other duties. The system administrator may even be you! Fear not, this book explains the chief responsibilities of the SA. For the moment, I will assume that you have someone who can get you online.

The first sign that your terminal awaits you is that your screen displays the *prompt*

```
login:
```

with the cursor blinking impatiently after the colon. A prompt is any message or symbol that indicates that input is expected—in other words, you are being *prompted* to type something! The cursor is a distinctive symbol on the screen indicating where your next typed character will appear.

Although you are online to Unix and Unix is talking to you, you are not yet an active user. You must first log in (some say log on) and satisfy Unix that you are a legitimate user. Let's see how.

LOGGING IN

In a single-user environment, such as DOS, anyone booting up the system can usually start operating it without permission. DOS itself just doesn't care. If you want to control access, you must install some additional security system (either mechanical locks or software barriers). With a multiuser system such as Unix, the need for security has to be taken more seriously since access may be possible from terminals beyond your immediate control. Unix offers a host of built-in security features that allow the system administrator to control who can get online as well as what the users can and cannot do when they get online. The system administrator is free to decide how much security is appropriate for a given installation, but there is a minimum level found at most sites.

The basic idea is that the system administrator assigns each user a unique *user name*, also called a *logon*, and an initial *password*. The SA sets up your logon and password on the system and tells you what they are. Once you have logged on (you'll see how soon), you are free to change your password but not your logon name. The latter is usually your first name in lowercase letters, such as `stan` or `mary`. In larger installations, it may be necessary to use fuller names and/or initials to ensure uniqueness. Whatever scheme is adopted, your logon *must* be in lowercase, and it serves to identify you to Unix at all times.

The choice of passwords needs careful consideration and we'll be discussing this in detail later on. For now, make sure you know your password and keep it to yourself. And try not to forget your password—it means more work for the system administrator, who may be annoyed enough to issue you a temporary, deflating password such as `dummy`.

At the `login:` prompt, type your logon. As you type, you'll see your keystrokes echoed on the screen. The stream of characters you type is actually going to a temporary storage area inside Unix known as a *buffer*. Each character in turn is sent back to the screen as you type—hence the terms *echo* and *echoing*. Most of your keyboard entries will be echoed in this way, but there are important exceptions, as you'll see shortly.

TYPING ERRORS

Your keyboard entry builds up in the buffer until you press `Enter`, at which point Unix starts examining what you have typed. You therefore have a chance to correct your input *before* you press the `Enter` key.

If you make a conscious typing error, there are two ways to recover: erase and retype characters or scrub the entire entry and start over again.

To erase characters to the left of your cursor, you can press the Backspace key. If your keyboard lacks a Backspace key, use `Ctrl-H`. `Ctrl-H` means holding down the `Ctrl` key and pressing the `H` key. You may see this written `^H` in Unix documentation.

To delete a whole line, type `Ctrl-U`, then retype from the beginning. (Your system may allow the use of `#` to delete the previous character, or `@` to delete the whole line. Check with your system administrator or experiment!)

THE ENTER KEY

When you think you have correctly typed your user name, press the Enter key. Until you press Enter, Unix will simply wait for more keystrokes. This is a general rule: pressing Enter signals the end of your typing and asks Unix to process what you have typed. Unix responds with the prompt

```
Password:
```

The password is case-sensitive. Carefully type your password exactly as the system administrator set it up (using the proper upper- and lowercase letters); then press Enter. (This is the last time I'll remind you that Enter is needed after every completed line of input.) I stress the word *carefully* for two reasons:

- ▶ What you type in the password field is *not* echoed on the screen, lest Peeping Toms are lurking behind you.
- ▶ Passwords are (or should be) strange, un-English, hard-to-type combinations.

Unix now checks both your logon name and your password. If the logon is valid and matches the password previously assigned, you will be logged into Unix.

LOGIN ERRORS

If you make a login error, you get the message

```
Login incorrect
```

followed by another login prompt:

```
login:
```

Note that Unix plays its cards pretty close here. If you type `stam` in place of `stan`, followed by Stan's password, Unix will not say `logon name invalid`. If you type `stan` followed by Mary's password, Unix will not say `password invalid`. All you know is that either the logon name, the password, or both do not match. This withholding of clues is all part of the cloak and dagger game—no help for the potential intruder. If you keep getting `Login incorrect` messages, check the Caps Lock key; you may be using the wrong case. If this doesn't help, you'll have to double-check with your system administrator.

LOGGED IN AT LAST!

Once you've passed the login hurdle, Unix can respond in many ways depending on the way your system is set up. Both you and the system administrator can tailor your environment to suit your lifestyle. This is the two-edged sword of Unix: infinite flexibility! In the simplest case, your successful login is greeted by a single character, known as the *shell prompt*. A \$ symbol indicates that you have the Bourne shell; a % means you have the C shell. (There are other shells and other prompts, but these two are the most common.)

The differences between the Bourne shell and C shell will not affect you until we reach more advanced Unix operations. The examples here display the Bourne shell, so when you see the \$ prompt in the following examples, make the mental adjustment to % if you are using the C shell.

I'll have more to say about shells later. For now, look on them as special Unix programs that will interpret your typed commands and pass them on for further action. A Unix shell is something like the command-line interpreter, COMMAND.COM, in DOS. Most Unix systems now offer you a choice of shells so that you or your system administrator can select the default shell to suit your tastes. The word *default* crops up frequently in data processing, so let's take a brief time-out to discuss what it means.

Defaults

If you have a simple situation where the computer expects a yes or no response, it may be that 90 percent of the time yes is the more appropriate response, as in `Are you sure you want to Exit? y/n?`. If the program has been arranged so that the Enter key works as though `y` had been entered, we say that `y` is the *default*. More generally, if one option among many is the one programmed to be selected in the absence of user input to the contrary, that option can be the default. In many Unix situations, you can change a default permanently or for a particular session. In the latter case, you would revert to the default default, as it were, in subsequent sessions.

Login Greetings

In addition to the shell prompt, Unix may reward you with a message or two. If Unix is uncertain as to your type of terminal, it may prompt you for information before proceeding:

```
TERM = unknown)
```

or

```
TERM = vt100)
```

In the latter case, the `vt100` (or some similar set of characters in parentheses) represents the default terminal type for your system. To pass this hurdle, you need to confirm this default name by pressing Enter or supply the actual terminal type. Your system administrator can help you here. Each terminal type supported by Unix has a mnemonic name such as `vt100` (the DEC VT-100), `wyse60` (for the Wyse 60), and so on. In most cases, the system administrator can set up the system so that the TERM message does not appear or simply requires you to press Enter.

The system administrator can also set up a message of the day (known as *motd* in Unix jargon) that will be displayed to all users when they log in. The message may be an important newscast (The system will be down next Friday) or a cheery Welcome aboard! My SCO UNIX also tells me useful facts about disk usage. You may see the message

```
You have mail.
```

This alerts you to the fact that the Unix electronic mail service has received one or more messages since the last time you cleared your incoming mailbox. Mail can originate from users on your system (you can even send yourself a letter) or from users on any other remote system that has your Unix mailing address and the right connections.

COMMANDS

What is the shell prompt prompting you for? The answer is Unix *commands*. These commands come in all shapes and sizes, from simple and common to rare and hellish. The command itself is a rather brief set of letters (usually, but not always, with a hint of mnemonicity), possibly followed by *options* and/or *arguments*. I'll explain these terms in the next chapter, but for now, let's try a few simple commands. I'll use boldface to show what you type and lighter type to indicate the response. First, enter the command `date`. The dialogue will look something like this:

```
$ date
Tue Feb 16 09:35:12 PST 1999
$ _
```

Note that after the date, time, and time zone are displayed, the shell prompt reappears ready for your next command. Next enter `who` to see

who is on the system. The response is a list of all active users, the terminals they are logged on, and the date and time they last logged on.

```
$ who
iwonka tty1a Feb16 10:05
stan   tty2a Feb16 15:23
$ _
```

Unix lets you enter two or more commands on the same line. Simply enter a semicolon between each command; then press Enter after the final command:

```
$ date; who
Tue Feb 16 09:35:12 PST 1999
iwonka tty1a Feb16 10:05
stan   tty2a Feb16 15:23
$ _
```

The next example shows what happens if you enter a nonexistent command.

```
$ amelia
amelia: not found
$ jimmy_hoffa
jimmy_hoffa: not found
$ _
```

No, Unix is not an expert in aviation and trade union history. If you enter a command that is unknown to Unix, you get the `not found` message, followed by the shell prompt.

It is important at this early stage to realize that it is almost impossible to damage a computer system by mistyping a command. There are, to be sure, a few dangerous commands, such as `rm` and `rmdir`, that if used rashly, can erase files you may not want to erase. I'll show you how to avoid such calamities, so there is no need to develop computophobia.

LOGGING OUT ALREADY?



WARNING

Be sure to log out after each Unix session.

After logging in and typing a few commands, it may seem premature to show you how to log out (some say log off). In fact, it is an essential operation to master as soon as possible. The whole point of security and passwords is lost if you leave your terminal in an active logged-in state,

whether remotely or locally. Anyone can take advantage of your hard-gained access, and who knows what evil lurks in the hearts of men and women? In addition, some sites charge you real money or set quotas for connect time. The logging out procedure varies according to the shell you are using. The Bourne shell logout works as follows:

```
$ exit
```

The C shell requires this command to log out:

```
% logout
```

At most sites, you can also log out with Ctrl-D. (Remember to hold down the Ctrl key and press D.) A successful logout is signaled by the appearance of the login prompt, so if you or anyone else wishes to become an active user, the full logging in sequence is required. If you are connected by modem, you must follow the vendor's instructions for hanging up. Many online Unix systems will hang up for you when you log out, but check this before you run up unnecessary phone bills.

CHOICE OF PASSWORDS

Passwords must strike a balance between memorability and lack of obviousness. These aims conflict somewhat. Passwords should be reasonably memorable; otherwise you'll have to write them down in too many accessible places. They should also be fairly long, say, six characters minimum. You have probably heard all the stories of hacker break-ins. Often the passwords were broken because the user used obvious words such as names of husbands, wives, dogs, or children. The SCO UNIX and other Unix systems offer the automatic generation of "pronounceable" but otherwise highly obscure passwords such as `k l i b r u g a k`, which you would remember in syllables as "kli-bru-gak." Another useful trick is to mix alphanumeric characters with punctuation or even invisible control characters.

WHAT'S NEXT

Chapter 2 introduces the Unix kernel, some of the more commonly used shells, and the basic organization of data within Unix: files directories and subdirectories. You'll also get to try out a few simple Unix commands.