Grassroots Series

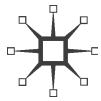
Mike Joy,
Stephen Jarvis
& Michael Luck

Introducing UNIX and Linux

Introducing UNIX and Linux

Mike Joy, Stephen Jarvis and Michael Luck

palgrave
macmillan



First published 2002 by
PALGRAVE MACMILLAN
Houndmills, Basingstoke, Hampshire RG21 6XS and
175 Fifth Avenue, New York, N. Y. 10010
Companies and representatives throughout the world

PALGRAVE MACMILLAN is the global academic imprint of the Palgrave Macmillan division of St. Martin's Press, LLC and of Palgrave Macmillan Ltd. Macmillan® is a registered trademark in the United States, United Kingdom and other countries. Palgrave is a registered trademark in the European Union and other countries.

ISBN 978-0-333-98763-6 ISBN 978-0-230-80245-2 (eBook)

DOI 10.1007/978-0-230-80245-2

This book is printed on paper suitable for recycling and made from fully managed and sustained forest sources.

A catalogue record for this book is available from the British Library.

10 9 8 7 6 5 4 3 2 1
11 10 09 08 07 06 05 04 03 02

Contents

Preface

UNIX is an operating system which has seen substantial growth in its popularity over the last few years and is used by many universities and colleges, as well as in industry. Linux is a UNIX-like operating system for PCs which is freely available and has become a serious alternative to proprietary systems such as Windows. This book is a *beginner's* guide for students who have to *use* UNIX and/or Linux. No prior knowledge of programming is assumed, nor is any experience of using computers. We do, however, expect our audience to have a serious interest in computing, and a typical reader might be a student in the first year of a degree or HND course.

UNIX is more than just a computer operating system: it is a philosophy of programming. Learning UNIX involves becoming familiar not only with the commands it affords the user, but also with the methodology it employs. It is a very powerful tool in the hands of an experienced practitioner, but it can be daunting for the novice. We introduce enough detail for the reader to be able to utilise the facilities in UNIX, but no more.

In 1993 an International Standard was published, known as 'POSIX.2', which specifies the constructs and commands that a UNIX system should have available to its users. This book follows that standard. However, POSIX is a 'minimal' standard, and most UNIX or Linux systems contain much more. We discuss in this book *all* the basic constructs and commands of UNIX (as defined in POSIX.2), sufficient for the reader to be able to use each of them, together with some of the more common and useful extensions. We do not delve into any in fine detail; part of the UNIX philosophy is that such information is available online. The reader who requires more sophisticated use of UNIX after reading this book will know how and where to find the extra information they need.

To get the most from this book, you should have access to a UNIX computer system or a PC running Linux, as much of the text relies on your being able to try out examples. If you have a PC running Windows, we discuss in Chapter 3 how you can install Linux on your PC.

This book is a new version of *Beginning UNIX*, which is no longer in print. The material covered in chapters 4 to 11 is substantially the same as the corresponding chapters in *Beginning UNIX*, but the remaining

chapters are new. We have expanded the coverage to include discussion of Linux and related issues, including installation and maintenance on a PC. A new chapter on Perl has been included. Technical material is now consistent with current Linux distributions in addition to Solaris and other versions of the UNIX operating system.

NOTE

Acknowledgements

Grateful thanks are due to Nathan Griffiths and Steve Matthews for commenting on draft versions of this book. Thanks also to Hugh Glaser for encouragement and feedback, and to students at Warwick and Southampton for valuable input.

NOTE

Trademarks

CHAPTER 1

The Computing Environment

CHAPTER OVERVIEW

This chapter

- reviews basic notions of computer hardware and software;
- outlines the different kinds of software program;
- introduces the basic philosophy of UNIX and Linux; and
- provides a brief description of the history of UNIX and Linux.

If you pick up any book over ten years old on the subject of computing, you could get quite different ideas of how people use their computers. The basic ways of using computers haven't changed, but modern computing places an unimagined amount of control and power with the individual user. This means that the user now has the ability (and quite often the need) to deal with issues relating to the administration of the computer to get the best out of it. In this book, we'll be explaining just how to understand what this involves, and how to minimise the amount of effort required for effective use of your computer.

We start in this chapter by reviewing some basic concepts of computing in a non-technical way, so that if you really are a beginner, reading through this chapter should bring you up to speed. If you are already familiar with the ideas of hardware and software, input and output, processors, systems software, and applications programs, you may choose instead to move swiftly on to Chapter 2.1, or simply to skim this chapter.

1.1 What is a Computer?

In very basic terms, there are essentially two kinds of "thing" involved in computing. There are things you can kick, actual bits of machinery that you can pick up and take away, including the computer itself, printers,

screens and other physical devices (digital cameras, scanners, disk drives, CD drives, etc.), which are collectively and individually known as **hardware** Thus, hardware includes the devices you use to communicate with a computer system (such as the mouse, keyboard), the actual components that make up that system, and any other devices.

Unfortunately, the hardware won't work by itself and needs detailed instructions, or **programs**, to make it do what it should. In addition to the hardware, therefore, it is also necessary to have a set of programs that tell the hardware what to do. These programs, which refer to the actual instructions rather than the medium on which they are stored, are collectively known as **software**. Software is needed for the basic operation of computers (like the software that is the subject of this book, UNIX and Linux) as well as for the more common applications that you may already be familiar with, such as word-processing, spreadsheets, games, MP3 playing, and limitless other possibilities. By themselves, hardware and software are not enough to do the things we want of computers — it is the combination of hardware and software that enables effective use of modern computers.

Below, we describe the different kinds of hardware and software in a little more detail.

1.2 Hardware

1.2.1 Processors

The most important part of the overall system is the **processor** (or **central processing unit**, **CPU**) on which the computer is based, and which does the main work of the system. In recent years, the advance of the PC has been fuelled by progress in such processors, which are becoming ever faster and more powerful. In PCs, these have included the series of Pentium processors developed by Intel, with alternatives from companies like AMD. Other computers have different processors, like Sun's SPARC processor. Whichever processor your machine uses is not important for now — there may be variations in speed and power, as well as in some other more technical differences, but the key point to note is that this is the main component of the machine.

ACRONYM

AMD = 'Advanced Micro Devices, Inc.'
SPARC = 'Scalable Processor ARChitecture'

1.2.2 Input Devices

Although the processor is most critical, it is of little use if you can't display the *results* of the computation it performs, or if you can't specify and modify the *kinds* of computation you want it to perform. For this reason, we need **input** and **output** devices — hardware components that allow users to interact with the processor in easy and convenient ways.

In order to instruct a computer to perform a task, we require a way to provide instructions as input. Perhaps the most recognisable input device is the **keyboard** (typically of the 'QWERTY' variety because of the layout of the keys, similar to a typewriter), which nearly all computers use to receive textual and numeric input. The keyboard is the most usual way in which people write programs or enter data on which those programs might operate. However, there are also many other ways to provide input to a computer. For example, many people now have scanners to enable graphical images to be provided as input. Similarly, digital cameras, bar-code readers in shops, and even sound recorders offer different ways of getting data to the processor. In this book, we will focus on the standard keyboard as our main input device, but we also note that the **mouse**, with the purpose of enabling the selection and movement of items displayed on the screen, is a vital part of modern computer systems.

1.2.3 Output Devices

Output devices are also varied, and we will focus primarily on the **screen** or **monitor** (or even **visual display unit** — **VDU** — to use a somewhat out-of-date expression) that typically comes as part of the package with the processor and the keyboard.

In the past, people used so-called **dumb terminals**, which are largely redundant now. A dumb terminal consists of a keyboard and a screen, and can display only the same sort of text and simple characters as a typewriter. On the screen is a **cursor**, which is either a block (a filled rectangle the size of a letter) or an underscore, marking the point on the screen at which typed characters appear, and also where any message the computer writes will begin. The equivalent of a dumb terminal is now more commonly referred to as a **command window** in many systems.

These days, modern computers typically use a screen (to display both the input that is provided through keyboards, for example, and any results of the processing performed), a keyboard and a mouse. The configuration is sometimes referred to as a **graphics terminals**, to distinguish it from a dumb terminal. These are capable of much more sophisticated output. In particular, the screen is a high-resolution display (nearly always colour), allowing complex graphics to be drawn as well as simple characters. Usually, a system is employed by which the screen is divided up into rectangular areas called **windows**, with which to communicate individually. The computer itself may be a PC, as shown in Figure 1.1, a **workstation** or a laptop, but that is not important. We will assume the use of a command window, which applies to all equally. If you are using any of these, you can create such a window, which behaves as if it were itself a dumb terminal, having its own cursor. Typically, there is also a global cursor that moves each time you move the mouse; you can

select which window the keyboard will communicate with by moving the global cursor so that it is within the chosen window.

The look and feel of the various kinds of computers can vary enormously, as can the way in which windows on a screen are manipulated. Either a **window manager** or a **desktop manager** can be used to control the windows on a screen, and though the distinction between the two kinds of software is somewhat blurred, a desktop manager typically has more features and capabilities than a window manager. We will go into more details about these later on in Chapter 3.

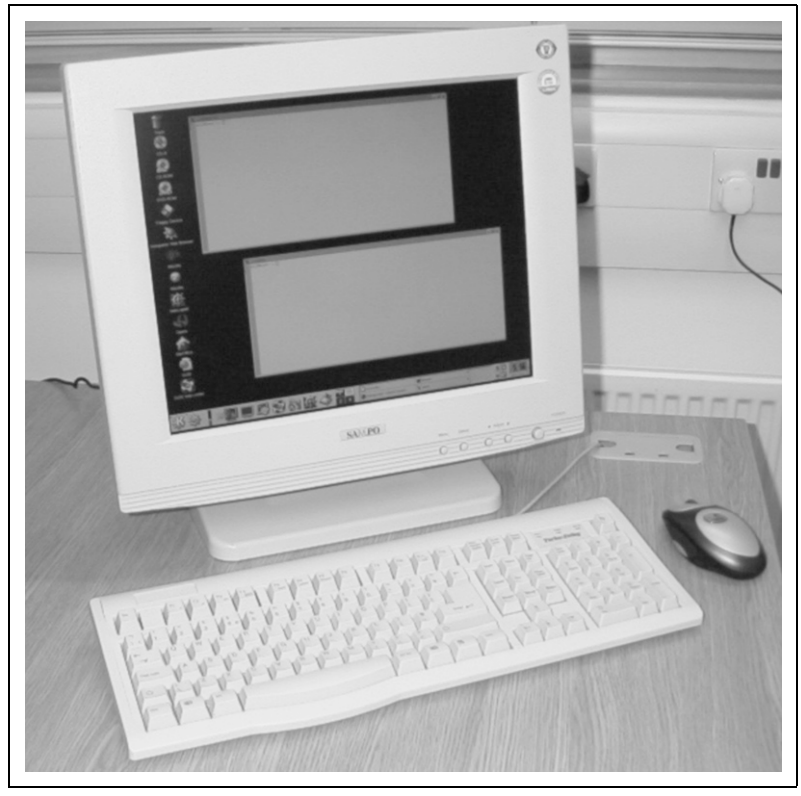


Figure 1.1 A typical computer, with screen, keyboard and mouse

These basic components of processor, screen and keyboard are the key pieces of the modern computing system, and their combination underlies all of the details that follow in this book.

1.3 Software

1.3.1 Input and Characters

When communicating with UNIX, users send to the system a **stream** of characters. Each time a key on the keyboard is pressed, a character is sent to the machine. Usually, the computer **echoes** the character so that it is

displayed on the screen. Similarly, to communicate with the user the system sends a stream of characters to the user's computer, which is able to interpret the particular character coding and display the characters on the screen accordingly.

While interacting with the system, a user types in lines of text from the keyboard, terminating each line by pressing the **Return** (or **Enter**) key. These lines are interpreted as instructions to UNIX, which then responds accordingly, and displays messages on the screen. On a graphics terminal, this dialogue is between the keyboard and a specific window (typically the window over which the cursor has been placed). Manipulation of other devices such as a mouse also results in the transmission of characters to the UNIX machine. However, these are interpreted as relating to the management and display of the windows only.

Commands sent to UNIX are executed by a program called the **shell**. We shall see later that the shell is just one example of a program that can run on a UNIX system, but is special since it is the principal interface between a user and the very heart of the system software, known as the **kernel**.

Most characters that we shall use are the **printing characters**. These, which include letters, digits, punctuation marks and the other symbols marked on the keyboard, are displayed in the obvious way. However, other characters, known as **control characters** (listed in Table 1.1), are sometimes required by a UNIX system. Each is stored as a number using a **character encoding** such as **ASCII**. For instance, the character whose code is 7 and is sometimes referred to as 'bell', if printed on your terminal, normally causes the terminal to make a noise (typically a 'beep').

ACRONYM

ASCII = 'American Standard Code for Information Interchange'

Each control character has a name, typically an acronym of its description. For character number 7, this is **BEL**, short for 'bell'. Control characters can be typed by pressing a key while holding down the **Ctrl** key. For BEL, this key is G, and for this reason BEL is often written as **ctrl-G** or ^G. Some of the other control characters have anachronistic names that also relate to the functioning of a teletype, but most of them will not concern us here.

NOTE

The bell character was originally used on 'teletype' terminals to attract the attention of the user in the days of the telegraph

Control characters have purposes which, for the most part, are obscure. Many are used by operating systems (not necessarily UNIX) to structure data, and have meanings with historical relevance only. Some of the more useful control characters include the following. The character **TAB** has the effect, when sent to the screen, of moving the cursor to the next tab position (usually columns 8, 16, 24, 32, etc.). The key marked TAB or →, when pressed, transmits a TAB character to the computer. The character **NL** (Newline) causes the cursor to move down to the left-hand side of the next row on the screen. This character is provided as input to the machine whenever the key marked RETURN (or ENTER or ↩) is pressed. The escape character, which would not normally be

Table 1.1 ASCII control characters

Code	ctrl-key	Name	Description
0	^@	NUL	null
1	^A	SOH	start of heading
2	^B	STX	start of text
3	^C	ETX	end of text
4	^D	EOT	end of transmission
5	^E	ENQ	enquiry
6	^F	ACK	acknowledge
7	^G	BEL	bell
8	^H	BS	backspace
9	^I	HT	horizontal tab
10	^J	NL	newline (linefeed)
11	^K	VT	vertical tab
12	^L	NP	new page (formfeed)
13	^M	CR	carriage return
14	^N	SO	shift out
15	^O	SI	shift in
16	^P	DLE	data link escape
17	^Q	DC1	device control 1
18	^R	DC2	device control 2
19	^S	DC3	device control 3
20	^T	DC4	device control 4
21	^U	NAK	negative acknowledgement
22	^V	SYN	synchronous idle
23	^W	ETB	end of transmission block
24	^X	CAN	cancel
25	^Y	EM	end of medium
26	^Z	SUB	substitute
27	^[	ESC	escape
28	^	FS	file separator
29	^]	GS	group separator
30	^^	RS	record separator
31	^_	US	unit separator
127	^?	DEL	delete

displayed on a screen at all, is sometimes required when typing in data. There should be a key on your keyboard marked ESC or ESCAPE.

1.3.2 Application Programs

As mentioned earlier, the hardware alone is not enough. To do anything useful with a computer, you need to run software, or programs, on the hardware. Programs can be used to do pretty much anything you want, but commonly include word-processing, scientific calculations and games, and even support the development of yet more programs. What is important to note here is that for each different application, you need a different **application program** to run on the computer. Thus, if you want to do some word-processing, you'll need to get a word-processing program to execute; word-processing can't be done directly by the computer otherwise.

Programming Languages

The processing units inside a computer understand a language called **machine code**, and all the calculations that a computer performs use this code. Machine code, which is a 'low-level' language, is specific to the particular make and model of computer on which it runs, and is not designed to be read by humans. Any instruction given to a computer must be translated (somehow) to machine code before the computer will understand it. It is unlikely you will ever need to come into direct contact with machine code.

Typically, programs are written in high-level languages that are easily readable by humans, but not by computers. They require **compilers** and **interpreters** to perform a translation into the machine code that computers can understand.

1.3.3 The Operating System

The final piece of the jigsaw of modern computing, to make the hardware and software work together, and to make the different bits of hardware like the screen, keyboard and processor talk to each other, is what is known as the **operating system**, which is **system software** as opposed to application software. The operating system is a complex program (or collection of programs) that controls the internal operation of the computer to ensure that all of the different things that are taking place at the same time are done effectively and sensibly. For example, while the computer accepts input from the keyboard and displays output on the screen, it may also be processing some data and accessing the hard disk, all at the same time.

Just as there can be different processors, different screens and keyboards, and different application programs, so there can be different

ACRONYM

DOS = 'Disk Operating System'
CPM = 'Control Program for Microcomputers'
VMS = 'Virtual Memory System'

operating systems. If you've got this far, then we should be able to assume that you know that your particular operating system is (or is likely to become) UNIX or Linux, but you may also be familiar with Microsoft's **Windows**, with **DOS**, or with some other operating systems such as **CPM**, **MacOS**, **Multics**, **BeOS**, **PalmOS** or **VMS**.

1.3.4 System Administration

Traditionally, UNIX systems have been multi-user systems with individuals simply gaining access as users. In these situations, there is usually someone, somewhere, who is in day-to-day charge of the system, and known as the **system administrator**. If you are using this kind of system and have problems that neither you nor your colleagues are able to resolve, then the system administrator will either be able to help, or at least point you in the direction of someone who can. You should find out who your system administrator is, and make sure that you are in possession of any documents that he or she wishes users of the system to have.

More recently, however, there has been a move towards the use of UNIX for individually-run personal computers, especially with the recent success of Linux. If this is your situation, then it is you who will act as the system administrator for your machine, and will be responsible for its maintenance. In particular, if you are using Linux on your own personal computer, make sure you read the handbook supplied with the operating system in conjunction with this book. If there are any differences, it will be an invaluable help.

Finally, there is one user of the system who is called the **super-user**. He or she has special privileges on the system, and is allowed to perform certain actions forbidden to ordinary users, such as having the unrestricted right to change and to delete files on the system. The super-user may or may not be the same person as the system administrator.

1.4 History of UNIX and Linux

ACRONYM

MIT = 'Massachusetts Institute of Technology'

The first UNIX system was built at **Bell Labs**, the research division of the US telephone corporation **AT&T**, in 1969. Prior to that date, Bell (together with **General Electric** and **MIT**) had been engaged in developing a large-scale operating system, known as 'Multics'. This collaboration between industry and the academic community had begun in 1964, but five years later it became clear that the three participants had different goals for the project. By this time a vast amount of work had gone into Multics, but more needed to be done for it to fulfil the aspirations of any of the participants. Accordingly, Bell Labs pulled out of the project.

Faced with not having a state-of-the-art operating system with which to work, a number of researchers at Bell, led by Ken Thompson and Dennis Ritchie, decided to create a new operating system 'from scratch'. Multics had become complex, and it was felt that a much simpler system was needed — the name 'UNIX' arose to emphasise that difference between it and Multics. The experience gained during the development of Multics contributed much to the design of UNIX.

A number of fundamental design decisions that were taken pervade the whole of UNIX. Programs written for UNIX should be simple, and should each do a single task well. This was different from the style adopted in some other operating systems, where large programs would be developed with many different capabilities, and would be commensurately complex. Also, programs should be designed so that they could easily be linked together, the output from one becoming the input to another. Thus it would be possible to build more complex programs by joining simple ones together.

Part of the philosophy underlying the design of UNIX was that the core system software, or kernel, should be as small as possible, and only perform those functions that are absolutely necessary — all other tasks should be the responsibility of the shell. At the same time as UNIX was being written, the language **C** was being designed, and in 1973 a UNIX kernel was written using C. C is a high-level language, and as such is machine-independent, so the new (small) kernel and shell could be transferred to a different machine easily. This was found to work well, and Bell Labs was happy to allow the source code for the kernel to be distributed to universities.

NOTE

The kernel is discussed in Chapter 2.1

In the next few years, work on UNIX was undertaken principally by Bell Labs and by the University of California at Berkeley. These two organisations, however, developed their own versions of UNIX, known respectively as **System V** and **BSD**. Industrial users tended to use System V, whereas BSD UNIX was common in universities and colleges.

ACRONYM

BSD = 'Berkeley System Distribution'

By the late 1980s UNIX had been implemented by many manufacturers, each of whom had developed versions which, although based either on System V or on BSD, had their own features. It became apparent that the popularity of UNIX, coupled with the proliferation of 'dialects', had resulted in a pressing need for a recognised standard for UNIX to be developed. This was taken on board by the **IEEE** under the name **POSIX**. POSIX consists of a number of interrelated standards. Now part of the **PASC** project, there are more than nine proposed POSIX standards, but not all are yet completed. In this book we only deal with POSIX.2, since the other standards are not necessary for understanding the UNIX shell.

ACRONYM

IEEE = 'Institute of Electrical and Electronics Engineers, Inc.'
PASC = 'Portable Application Standards Committee'

In 1991, a computer science student at the University of Helsinki in Finland, Linus Torvalds, decided to create his own version of UNIX, which he named Linux. It was in 1994 that he released version 1.0 of

Linux. Very quickly it became clear that Torvalds alone would not be able to develop a complete operating system, so he chose to open up his project to allow others to contribute to its development. On the Internet, Torvalds announced his project and called for volunteers to assist; in doing so, the source code was made freely available.

As a result of this model of allowing developers from around the world to contribute to the development of Linux, a Linux community was born, and has now grown to millions of users, numerous different Linux distributions, and over a hundred developers just for the Linux kernel. It is now an effective and successful operating system that competes on many platforms with commercial offerings. The latest version at the time of writing is version 2.4.

1.5 Conventions

Several different fonts are used in this book. **Bold face** is used when names or concepts are first introduced, and occasionally for emphasis. When dialogue with a machine is displayed, `fixed width font` is used for messages that the UNIX system prints, and **`(bold) keyboard font`** for instructions typed by a user. If a word that would normally appear in such a dialogue appears in the text, `fixed width font` is again used.

For most purposes in the book, the words 'UNIX' and 'Linux' are interchangeable, and unless otherwise stated use of the word 'UNIX' should be understood as meaning 'UNIX or Linux'.

CHAPTER SUMMARY

- Modern computer systems are made up of both hardware and software.
- Hardware comprises processors, and input and output devices.
- Software can be application programs or system software like operating systems.
- UNIX and Linux are operating systems with a long academic tradition.

CHAPTER 2

UNIX and Linux Design and Organisation

CHAPTER OVERVIEW

This chapter

- introduces the basic organisation of UNIX and Linux;
- describes the key underlying concepts;
- outlines the basic technical components necessary to get started; and
- gives details of how to get a copy of Linux.

2.1 The Kernel and Shell

In order for a computer to do any useful work, it must also perform 'housekeeping'. It needs to understand that it has various devices such as printers connected to it, and it needs to know when a user wants to run a program. These tasks, are performed by an **operating system**, together with many others that are required for the computer to function effectively, but are not of interest to the user. An operating system is a program, or collection of programs, that runs whenever the computer is switched on. It controls the computer, allows the user to type in instructions to the computer, and performs many other necessary functions. UNIX is an operating system.

A UNIX system can be split into two parts. While the system is operational, a program called the **kernel** is constantly running. This is what forms the core of the operating system and is central to UNIX. In

this book, we will not be concerned with how the kernel functions, since it is not information which the user needs to know.

The other part of a UNIX system is a **shell**, which is the interface between a user and the system itself. It allows the user to instruct the machine and to run programs. A shell communicates with the kernel, but keeps the user at arm's length from it, as illustrated in Figure 2.1. In order to use a UNIX system, it is sufficient to understand a shell; the kernel can remain hidden from the user.

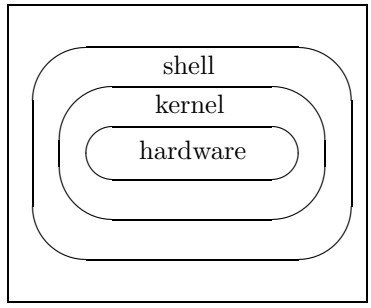


Figure 2.1 The UNIX kernel and shell

The kernel is always present, but the shell is only active when someone is actually using the UNIX system. Since the shell enables the user to instruct the system to perform tasks, the instructions that can be given to the shell must be easy for a person to understand. Different individuals have had different ideas about exactly how a shell should be designed, and a number of different shells have been devised in the past. They are all similar to each other, but differ in details. The first shell, historically, is the **Bourne shell**, known as **sh** and named after its creator. This shell is still used today, although newer shells with more powerful features have been created which are effectively extensions of the Bourne shell. These include the **Korn shell** (**ksh**), the **Z shell** (**zsh**), and **bash**. A programmer familiar with the Bourne shell should have no trouble using any of these three other shells. Indeed, if such a programmer were not using the extra features provided by these shells, he or she would be unaware that the shell was not the Bourne shell.

ACRONYM

bash = 'Bourne Again SHell'

The **C shell**, known as **csh**, has a syntax that resembles that of the C programming language, and is markedly different from any of the shells based on the Bourne shell. A programmer familiar with the Bourne shell would not be able to use the C shell without learning the differences between it and the Bourne shell. Just as there are shells that are extensions of the Bourne shell, so the C shell itself has been developed into shells with extra facilities. The most common of these is the the **tcsh** (pronounced 'teesh').

POSIX.2 defines the 'standard' shell, and is modelled principally on the Bourne shell. The POSIX.2 shell contains features that have been added to the Bourne shell in the light of experience gained with other

shells. Much of what is discussed in this book will thus be true for the Bourne shell. It is likely that as existing shells derived from the Bourne shell, such as ksh and zsh, are developed, each will be amended so that its specification conforms to POSIX.2.

2.2 Files

On any machine there will be a large amount of information (or data) that must be stored, including programs, text, and the UNIX operating system itself. Each unit of data — which may be small (for instance, a few words of text) or large (like parts of the UNIX operating system itself) — is stored in a file. Files are simply sequences of bytes, stored somewhere on the system, perhaps on magnetic disks, CD-ROMs, or other storage devices. We are not interested in exactly where the file is stored, merely in its contents.

Each file has a name, which should consist of any letter, digit, or the characters `.` (period), `-` (minus sign), or `_` (underscore). Other characters are also acceptable in a filename, but are discouraged in order to promote clarity. When we use files, we will normally refer to them by name. Some examples are:

```
test    11a    My_File    prog.c    p-1
```

2.2.1 Networks

ACRONYM

CPU = 'Central Processing Unit'

Computer systems contain at least one computer. However, it is becoming increasingly difficult to define what is meant by 'a computer' — until a few years ago, a computer would have had a single CPU, which would perform all the computational tasks.

Nowadays, a computer may contain several processing units around which the workload will be distributed. In addition, several computers may be connected together in a **network** where each constituent computer can communicate with others in the network.

In some cases, the computers in a network are very intimately connected, and the network appears to a user as a single but very large computer. We use the word **system** to mean either a single computer or a network of computers that appear to the user as a single entity. A campus-wide UNIX network would be an example of such a system; a more loosely-connected network such as the Internet would not be. When using a terminal on a network, users are still communicating with a specific machine. Each window allows a dialogue with a single UNIX machine, and it is that target UNIX machine with which we shall be concerned in this book.

2.3 Technical Basics

2.3.1 Bits, Bytes, Words and Characters

Data inside a computer is stored as a sequence of binary digits. Each such digit is called a **bit**. Exactly how bits are stored does not concern us here, but several different methods can be used depending where on the computer system the data is required. Bits are grouped together in groups of (usually) 8 to form a **byte**. Bytes are then grouped in 2s, 4s or 8s to form **words**, the number of bytes in a word depending on the machine being used.

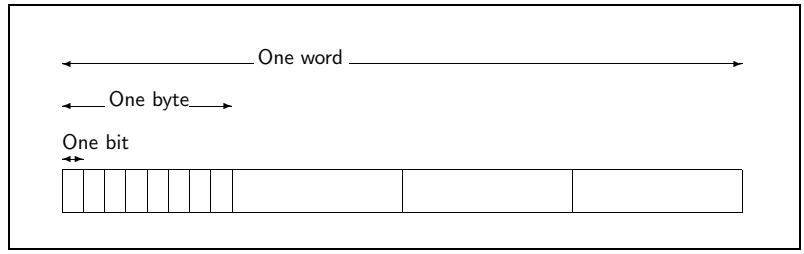


Figure 2.2 A 4-byte word

It is rarely necessary to enquire what individual bits are stored on a computer. Normally, the byte is regarded as the most basic unit of storage on a machine. Since a byte contains 256 permutations of eight binary digits, a byte can represent any number between 0 and 255 inclusive (or between −128 and +127, or other such ranges).

Just as with a typewriter, communication with UNIX is character-by-character. Unless you are dealing *bit-by-bit* with the data stored in the system's memory, it is helpful to think of each byte representing a character, such as the letter 'A' or the symbol '@', since there is a correspondence between characters and the numeric codes (between 0 and 255) that can be stored in a byte. The most common coding scheme used is called **ASCII**, in which codes for the upper-case letters 'A' to 'Z' are 65 to 90, for lower-case letters 'a' to 'z' they are 97 to 122, and for the digits '0' to '9' they are 48 to 57. Other codes are used for other symbols. The codes are summarised in Table 2.1.

ACRONYM

ASCII = 'American Standard Code for Information Interchange'

In the earlier days of computing, the electronic components were often unreliable, and the final bit in a byte was used as a **check digit** whose value is determined by a simple calculation from the other seven bits. If one of the other seven bits is changed, the value of the eighth, which is referred to as a **parity bit**, is also changed. This parity check can then be used to identify bytes whose contents have been accidentally altered.

Table 2.1 ASCII characters

Code		Description
0-31		control characters (see Table 1.1)
32		space
33	!	exclamation mark
34	”	double quote
35	#	hash
36	$	dollar
37	%	percent
38	&	ampersand
39	’	single quote
40	(	left parenthesis
41	)	right parenthesis
42	*	asterisk
43	+	plus
44	,	comma
45	-	minus
46	.	dot
47	/	slash
48-57	0-9	digits
58	:	colon
59	;	semicolon
60	<	less than
61	=	equals
62	>	greater than
63	?	question mark
64	@	at
65-90	A-Z	capital letters
91	[	left bracket
92	\	backslash
93	]	right bracket
94	^	caret
95	_	underscore
96	‘	backquote
97-122	a-z	lower case letters
123	{	left brace
124	|	bar
125	}	right brace
126	~	tilde
127	DEL	delete (control character)

Parity checking is an unsophisticated form of error detection, and modern equipment seldom uses it, thus allowing 256 character codes to be stored in a single 8-bit byte, rather than just 128. Usually the first 128 match the ASCII character set, and the remaining characters are used for extra symbols, such as currency symbols and accented letters from languages other than English. One such code is known as **LATIN-1**. For the symbols used in this book these two codings are identical. Other codings do exist, however, perhaps the best known being **EBCDIC** and the 16-bit **Unicode**, but for the purposes of this book, we shall assume that ASCII is being used.

ACRONYM

EBCDIC = 'Extended Binary Coded Decimal Interchange Code'

Note that if you total the number of letters, digits, punctuation marks and other graphics symbols, there are nowhere near 256 of them — some codes relate to *non-printing* characters. These are characters which, rather than representing a symbol that can be printed on a computer screen, denote other actions that the computer display can perform.

2.4 How to get Linux

For many years there has been a tradition in universities of freedom of information, and results of academic research are typically easily accessible. Furthermore, software created during that research is often made available free of charge, either as public domain (where copyright no longer applies) or as shareware (where, although copyright still applies, the copyright owner permits copying). The source code for that software may also be available, and much software is now **open source**, where the source code is distributed and the user licence prohibits sale of the software without the source code. Linux is open source.

You may have access to a UNIX system via your university or college. If you don't, or you would like to use UNIX on your PC at home, Chapter 3 tells you how to get and install your own copy of Linux.

CHAPTER SUMMARY

- The kernel is the core of the UNIX operating system.
- The shell is the interface between the kernel and the user.
- Data inside a computer is organised in bits, bytes, words, characters and files.
- There are several different shells and character codings.

CHAPTER 3

Installing Linux

CHAPTER OVERVIEW

This chapter

- shows you how to collect system information about your computer;
- introduces you to the different options for setting up Linux;
- provides guidelines on how to install Linux; and
- highlights some of the everyday features which Linux will provide.

The purpose of this chapter is to arm you with the necessary information to install your own version of Linux. There are a number of Linux configurations you might consider and the choice you make will be influenced by the capabilities of your computer. This chapter is designed to help you recognise these choices and their limitations.

3.1 Starting out

If you want to download a freely distributed version of Linux, you can do no better than starting your venture at `www.linux.org`. As well as documenting general Linux information, details of the various Linux applications and on-line Linux tutorials, this official web site also plays host to the distribution of the numerous Linux packages.

The distribution of Linux is now extremely well supported and you will find that there are references to English and non-English language versions. There are also links to mirror sites where you can download Linux free of charge; there are details of Linux vendors and on-line reviews of some of the free Linux distributions.

At the last count there were at least 40 versions of Linux which could be downloaded from this site. This chapter does not deal with the specifics involved in downloading any one of these distributions, but you

will find that the on-line documentation provided with each package is quite adequate. However, there are a number of fundamental choices of which you should be aware when installing Linux; it is to these that we turn our attention in this chapter.

3.2 Preliminaries

Before you begin with the installation of Linux it is important that you establish a match between the requirements of your chosen Linux download and the capabilities of your computer system.

Linux turns out to be extremely versatile and it is therefore likely that your computer will be able to run Linux in one form or another. The 'build' you choose may, however, depend on the amount of processor power you have available, the amount of **RAM** and the amount of hard disk space you are able to commit to Linux.

ACRONYM

RAM = 'Random Access Memory'

3.2.1 Collecting information about your system

If your computer is already running a version of the Microsoft Windows operating system, then you can find information about your system by clicking on the 'Start' button and selecting 'Settings' and the 'Control Panel'. From here you should click on the 'System' icon; this will bring up a window entitled 'System Properties'.

The easiest way to capture the information that the 'System Properties' windows provide is to print a System Resource Report. You can do this by selecting the system properties 'Device Manager' window and then clicking 'Print'. At the print menu you should select the 'All devices and system summary' option and then press 'OK'.

If your computer is still using one of the older versions of Windows (3.1, 3.2, etc.) then you can gather and print the same system information by running the MSD.EXE utility from a DOS command prompt.

Among the information provided with each of the Linux distributions will be listed the minimum system requirements. Before you decide on which version of Linux to install, it is worth making sure that your System Resource Report meets these requirements.

3.2.2 Installation options

A second issue likely to affect the type of Linux distribution you choose is the way in which you intend to use Linux on a day-to-day basis. You might want to install the Linux operating system as the sole operating system on your computer. If this is the case then you should probably install Linux as a 'single boot'. This means that when you turn your computer on it only recognises the one operating system.

If, however, you want to retain the use of your existing operating system, for example, you would like to be able to run Linux *or* Windows, then you should choose a '**dual boot**' option. This means that the computer is aware of two different operating systems when it is turned on. It is also possible to run one operating system inside another with the aid of an **emulator**.

The next sections provide some of the detail which you will need to be able to choose between installation options. Note the pros (+) and cons (-) of each method; you should also be aware that the installations have very different hardware requirements and that your choice may to some extent be determined by the capabilities of your computer.

3.3 Single boot

NOTE

(+) reliable install
(+) less disk and processor
(–) can only run one OS

If you are installing Linux on an old computer (without a CD drive, with 32 Mb of memory or less, or with a 75 MHz processor or thereabouts) then you will find that the way in which you install Linux is already limited. In this case you must install Linux as a 'single boot' system. This means that you must essentially forfeit your previous operating system for your new version of Linux. While this is not always what people want (as you may still want to use Windows from time to time), this is the simplest way to install Linux on your machine.

ACRONYM

FDISK = 'Fixed disk utility'

You need to partition your hard disk before Linux can be installed. It is therefore worth checking whether the version of Linux you have chosen has its own partitioning software as part of the software bundle. If not, you will have to use the DOS/Windows **FDISK** program.

The single boot option does have a number of benefits. For example, it provides a fast, reliable and easy to use system.

3.4 Dual boot

If you possess a more up-to-date computer, or you are keen not to lose the use of your previous operating system, then you should install Linux as a 'dual boot' system. This has the overwhelming advantage of allowing you to switch between operating systems (Windows and Linux for example) and being able to use the applications provided by each. There are a number of possible dual-boot set-up configurations, which we consider next.

3.4.1 Booting from CD/floppy

NOTE

(+) minimal install
(–) not as fast as a hard disk boot

Many of the Linux packages come with a boot floppy disk. If you download Linux free of charge then you can create your own boot floppy from the download; you can also order these disks online or purchase a package with a free distribution copy inside.

The advantage of this approach is that you only need to do a minimal installation (of approximately 150 MB) for your system to be Linux usable. The disadvantage is that you need to keep your boot disk handy and may require your CD or floppy for other purposes.

3.4.2 Booting from your hard disk

NOTE

(+) faster than booting from floppy
(–) requires maximum install disk space

Dual booting Linux from the hard disk is a popular option. It allows you to select which of your two (or more) operating systems you wish to use when you boot-up your computer and it will leave any CD or disk drives free for other use. Although this mode of working does not allow you to run both operating systems simultaneously (see the Section 3.5 on emulators if this is your requirement) it allows you to maximise the speed at which both operating systems are able to co-exist and run independently on your computer.

3.4.3 A partitionless install

NOTE

(+) avoids disk partitioning
(–) may impact on existing OS

It is possible to provide a dual boot system without any repartitioning of your hard disk. However, repartitioning is a way of keeping the file systems and operating components of your two operating systems completely separate, see below. As a result you may find that you achieve a more reliable build if a dedicated Linux partition is provided.

If you are aiming to set up a dual-boot Linux on a non-partitioned disk — an option which is often offered with many of the Linux distributions — then you should search for a distribution of Linux that uses **UMSDOS**. This allows Linux and DOS to coexist in the same partition and uses the Linux loader **loadlin** to boot between each.

ACRONYM

UMSDOS = 'UNIX under MS-DOS'
GUI = 'Graphical User Interface'

While this might seem like a good solution, it should be noted that a partitionless dual-boot installation may have serious implications for your existing operating system. Rather than take this risk, and to be completely sure that you achieve a clean installation, it is better to opt for a dedicated Linux partition.

3.4.4 A dedicated Linux partition

NOTE

(+) most installers automatically partition
(+) little impact on existing OS
(–) must clear proposed partition
(–) must calculate partition size

If you have enough disk capacity, then you can allocate a complete partition (or indeed disk) to Linux. If you are serious about running a safe, clean and reliable Linux alongside an operating system such as Windows, then this is a good option. Recognising this fact, many of the Linux installers provide a GUI-based partitioning package that is fairly simple to use. Partitioning is discussed in more detail in Section 3.6.2 below.

3.5 Emulators

Operating system emulation allows you to run multiple operating systems concurrently without having to reboot. At best it allows you to run a full version of Windows inside Linux through the creation of something called a **virtual computer** — see subsection 3.5.1 below on VMware.

Alternative solutions include those systems that allow Windows applications to run under Linux without modification. Although this is not regarded as true emulation, it does provide a considerable level of Linux and Windows compatibility — see subsection 3.5.2 below on WINE.

3.5.1 VMware

NOTE

(+) runs Windows and Linux simultaneously
(+) unrivalled capabilities
(–) speed hit on guest OS
(–) costs around $300 at the time of writing
(–) "commercial software"

VMware is a commercial software package that allows you to run more than one operating system simultaneously. This is done by setting up a *host* operating system and one (or more) *guest* operating systems, each of which runs in an unmodified state.

Each guest operating system runs in a secure virtual machine; the beauty of VMware is that when using these virtual machines it is as easy to swap between operating systems as it is to swap between windows. In fact, if you run the virtual machine window in full screen mode, it is as if the guest operating system (OS) is the only OS on your machine.

Such sophistication must come at a price. Firstly, you will need a machine with a bare minimum of 256 MB of RAM and a 400 MHz processor. You will also need at least 500 MB of disk for the guest OS and the associated applications. Secondly, there will be a speed reduction when using the guest OS. This can be as much as 50%, which may be a problem if your machine is at the bottom end of the hardware requirements.

VMware is very well supported and you can configure your system so that the host operating system is chosen from any of Windows XP, 2000 and NT, Red Hat Linux, SuSE Linux and Mandrake Linux; the guest operating systems include all of the above and also the Windows 3.x/9x series.

You can find out more about VMware at `www.vmware.com`.

3.5.2 WINE

ACRONYM

WINE = 'Wine Is Not an Emulator'

WINE allows most Windows applications to be run natively under Intel versions of UNIX. WINE does this by providing low-level compatibility for Windows programs running under Linux. As a result, the applications run faster than they will under an emulator.

One of the main reasons for choosing WINE over an emulator is that it does not require extensive hardware resources. If you have a processor sufficiently powerful to run Linux then you will be able to run both

WINE and Microsoft Windows applications under it. As far as disk space is concerned, you only need approximately 250 MB of free disk to be able to store and compile the source code plus an additional 18 MB of `/tmp` space (see below) and 50 MB of disk in order to do the install. Another advantage of WINE is that it is free.

NOTE

(+) WINE is free
(+) applications run fast
(–) some applications don't work

Corel has been using WINE to port its WordPerfect Suite to Linux, so there are some well-documented success stories. However, there are difficulties with some of the Windows applications, so it is worth looking at the Application Database on the WINE web page before deciding whether this is going to be appropriate for your needs.

Once you have set up WINE on your computer, you can install Microsoft applications by invoking a terminal window and typing `wine` followed by the name of the set-up program. A similar procedure is used to run the application once it has been installed.

More details on WINE can be found at `www.winehq.com`.

3.6 Installing Linux

Once you are satisfied that you have chosen an appropriate version of Linux that matches the capability of your computer and also meets your own needs, you are ready to begin the installation process.

3.6.1 Installer software

ACRONYM

`YaST` = *'Yet another Setup Tool'*

Most of the Linux distributions, including SuSE and Red Hat, have very good installer software. The SuSE distribution includes the text-mode YaST installer which is designed to make the process of installation as painless as possible. You may also find references to the GUI-based YaST2, which, despite being more memory intensive, is easier to use. Each version of Linux has its own tool similar to YaST.

The installer software will probably provide you with a number of installation modes such as **recommended**, **customized** and **expert**. If this is your first Linux installation then you should choose a *recommended* installation. This will automatically install the core components and yet provide you with enough options to maintain control over the amount of memory needed for the installation.

Many of the installation questions are straightforward. The choice is less clear, however, when you are asked whether you are installing as a **workstation**, as a **server (installation)** or for **development**. Again, if this is your first installation and you are planning on using your computer as a stand-alone machine, then you should opt for the 'workstation' mode. You should also select a security setting if prompted to do so; something around 'medium risk' should be adequate if you are planning on connecting your computer to the Internet.