Linux Installation and Getting Started

Version 1.1, 25 September 1993.

This book is an introduction to the Linux system, including information on how and where to get Linux, installation of the software, a beginning UNIX tutorial, and an introduction to system administration. This is the first book anyone with interest in Linux should read.

Contents

Preface

It's unfortunate that the Intel 80386 and 80486 line of processors has been subjected to the growing popularity of so-called operating systems such as MS-DOS, an aging system which doesn't exploit the power of these machines. It's also unfortunate that better operating systems, commercial UNIXes and OS/2 included, are so large and expensive—not the enthusiast's forte. The question everyone's been asking is, "Who's going to save the IBM PC?" Who *is* going to save the PC world from the death grip of commercial operating systems? Where's Richard Stallman when you need him?

Things were looking extremely bleak, as the dark cloud of Microsoft and IBM loomed menacingly on the horizon. But, at long last, a messiah emerged unexpectedly from the fjords of Helsinki. Kernel in hand, with a growing band of programmers, beta testers, and other such brigands in tow—Linus Torvalds. And his system was known as Linux: the free UNIX clone for the 386.

The author hopes that this book will bring the masses the godfoot of that most miraculous of systems. Until recently only those with a true aptitude for UNIX and, indeed, computers in general have been able to take the plunge and run Linux on their machines. We have braved the jungle of conflicting and obsolete Linux documentation, and have fought with ardor to clear the path for newcomers, those without the alleged software machete. But trailblazing can only go so far. Sometimes it's best to burn down the entire jungle and start anew. This manual is the result of this destruction and rebirth.

Finally, the cowering bourgeois can install and run Linux on almost any 386 or 486 based machine. Even those who have never seen or touched the likes of UNIX before can immerse themselves in the beauty of true multitasking, X Windows, TEX, and a myriad of GNU software. And it doesn't cost a shilling—it is the free software buff's dream. It is the chance for all of us UNIX-developer wannabe's to get our hands dirty with kernel programming, porting software, writing documentation, and fixing bugs. It is the frustrated programmer's recluse from segmented memory and false 32-bit environments. It is the neophyte's escape from MS-DOS to the Elysian Fields of UNIX, the future.

Audience

This book is for any personal computer user who wants to install and use Linux on their system. We assume that you have basic knowledge about personal computers and operating systems such as MS-DOS. No previous knowledge about Linux or UNIX is assumed.

Organization

This book contains the following chapters.

Chapter 1, *Introduction*, gives a general introduction to what Linux is, what it can do for you, and what is required to run it on your system.

Chapter 2, *Getting Linux*, explains how to obtain the Linux software, either via U.S. Mail or from the Internet, via FTP. This discussion focuses on the SLS distribution of Linux.

Chapter 3, *Installing Linux*, explains how to install the Linux software, from repartitioning your drive, creating filesystems, and loading the software on the system. Again, this discussion focuses on the SLS distribution.

Chapter 4, *Linux Tutorial*, is a complete introduction to using the Linux system for UNIX novices. If you have previous UNIX experience, most of this material should be familiar.

Chapter 5, *System Administration*, introduces many of the important concepts of system administration under Linux. This will also be of interest to UNIX system administrators who want to know about the Linux-specific issues of running a system.

Chapter 6, *Advanced Features*, introduces the reader to a number of advanced features supported by Linux, such as the X Window System and TCP/IP networking.

Appendix A, *Tips, Tricks, and Common Problems*, contains other information of interest to those setting up a new Linux system, such as how to install Linux from the hard drive, and how to set up Linux for use with multiple filesystems.

Appendix B, *Bibliography and Sources of Information*, is a listing of other sources of information about Linux, including newsgroups, mailing lists, online documents, and books.

Appendix C, *Quick FTP Tutorial*, is a tutorial for downloading files from the Internet with FTP. This appendix also includes a listing of FTP archive sites which carry Linux software.

Appendix D, *Linux BBS List*, is a listing of bulletin board systems worldwide which carry Linux software. Because most Linux users are do not have access to the Internet, it is important that information on BBS systems becomes available.

Appendix E, *The GNU General Public License*, contains a copy of the GNU GPL, the license agreement under which Linux is distributed. It is very important that Linux users understand the GPL; many disagreements over the terms of the GPL have been raised in recent months.

Acknowledgments

This book has been long in the making, and many people are responsible for the outcome. In particular, I would like to thank Larry Greenfield and Karl Fogel for their work on the first version of Chapter 4, and to Lars Wirzenius for his work on Chapter 5. Thanks to Michael K. Johnson for his assistance with the LDP and the LaTeX conventions used in this manual. I would also like to thank Andy Oram, Lar Kaufman, and Bill Hahn at O'Reilly and Associates for their assistance and

interest in the Linux Documentation Project. And, of course, a special thanks to the many activists, especially Linus Torvalds and Peter MacDonald, without whom this would not have been possible.

In the last two years, we've seen Linux grow from a single hacker's operating systems programming project into a fully-functional UNIX clone, with an expanding support base of software, users, and programmers. Linux, as it stands now, is one of the greatest accomplishments of free software in existence, and I can only anticipate its further growth. I'd like to call it a revolution. I guess we'll see. So, gentle reader, I give you Linux—soon to be your faithful companion in the war against the wrath of proprietary operating systems. Take no prisoners.

Matt Welsh
5 August 1993

Credits and Legalese

The Linux Documentation Project is a loose team of writers, proofreaders, and editors who are working on a set of definitive Linux manuals. The overall coordinator of the project is Matt Welsh, aided by Lars Wirzenius and Michael K. Johnson.

This manual is but one in a set of several being distributed by the Linux Documentation Project, including a Linux User's Guide, System Administrator's Guide, and Kernel Hacker's Guide. These manuals are all available in LaTeX source format and Postscript output for anonymous FTP from `tsx-11.mit.edu`, in the directory `/pub/linux/ALPHA/LDP`.

We encourage anyone with a penchant for writing or editing to join us in improving Linux documentation. If you have Internet e-mail access, you can join the `DOC` channel of the `Linux-Activists` mailing list by sending mail to

```
linux-activists-request@niksula.hut.fi
```

with the line

```
X-Mn-Admin:  join DOC
```

as the first line of the message body.

Feel free to get in touch with the author and coordinator of this manual if you have questions, postcards, money, or ideas. Matt Welsh can be reached via Internet e-mail at `mdw@sunsite.unc.edu`, and in real life at

205 Gray Street
Wilson, N.C. 27893
U.S.A.

205 Gray Street NE, Wilson NC, 27893 USA
`mdw@sunsite.unc.edu`

Documentation Conventions

These conventions should be obvious, but we'll include them here for the pedantic.

Bold — Used to mark **new concepts**, **WARNINGS**, and **keywords** in a language.

italics — Used for *emphasis* in text, and occasionally for quotes or introductions at the beginning of a section. Also used to indicate commands for the user to type when showing screen interaction (see below).

⟨*slanted*⟩ — Used to mark **meta-variables** in the text, especially in representations of the command line. For example,

> `ls -l` ⟨*foo*⟩

where ⟨*foo*⟩ would "stand for" a filename, such as `/bin/cp`.

`Typewriter` — Used to represent screen interaction, as in

```
$ ls -l /bin/cp
-rwxr-xr-x  1 root    wheel    12104 Sep 25 15:53 /bin/cp
```

Also used for code examples, whether it is C code, a shell script, or something else, and to display general files, such as configuration files. When necessary for clarity's sake, these examples or figures will be enclosed in thin boxes.

Key — Represents a key to press. You will often see it in this form:

> Press return to continue.

◇ — A diamond in the margin, like a black diamond on a ski hill, marks "danger" or "caution." Read paragraphs marked this way carefully.

Chapter 1

Introduction

This book is about Linux, the free clone of the UNIX operating system for 80386 and 80486 machines. The rationale for writing this manual is clear: as Linux has grown and gained popularity, the old approach of searching a myriad of old documentation and software, downloading megabytes of files, and hoping that everything works is no longer adequate. Surprisingly, the number of MS-DOS and OS/2 users who are moving to Linux is large, and many of these users are new to UNIX. Here, rolled into one book, is a complete guide to installing Linux, with enough introductory material on using UNIX to get new users on their feet.

Linux is a clone of the UNIX operating system that runs on Intel 80386 and 80486 based machines. It supports a wide range of software, from TEX to X Windows to the GNU C/C++ compiler to TCP/IP. It's a versatile, bona fide implementation of UNIX, freely distributed by the terms of the GNU General Public License (see Appendix E). Before we delve any further, a few words about this book.

1.1 About This Book

In this manual, we'll cover:

- What Linux is;
- How to obtain and install Linux on your system;
- Getting started with Linux, even if you've never UNIX before;
- The basics of Linux system administration;
- An introduction to upgrading and installing new software.

This manual assumes no previous knowledge of Linux or UNIX. Some experience with Intel-based personal computers is assumed, and familiarity with MS-DOS or Microsoft Windows is helpful.

If you have previous UNIX experience, you may wish to skip ahead to Chapter 2, "Getting Linux," and then read through Chapter 3, "Installing Linux," and from there you're set. You may also wish to read Chapter 6, discussing some of the more advanced and interesting features of Linux. The rest of the manual covers issues of concern to those new to UNIX.

1.2 Introduction to Linux and UNIX

UNIX is one of the most popular operating systems worldwide because of its large support base and distribution. It was originally developed as a multitasking system for minicomputers and mainframes in the mid-1970's, but has since grown to become one of the most widely used operating systems anywhere, despite its sometimes confusing interface and lack of central standardization.

The real reason for UNIX's popularity? Many hackers feel that UNIX is the Right Thing—the One True Operating System. Hence, the development of Linux by an expanding group of UNIX hackers who want to get their hands dirty with their own system. Doug Gwyn said, "UNIX was never designed to keep people from doing stupid things, because that policy would also keep them from doing clever things." Most UNIX hackers would agree.

Versions of UNIX exist for many systems—ranging from personal computers to to supercomputers such as the Cray Y-MP. Most versions of UNIX for personal computers are quite expensive and cumbersome. At the time of this writing, a one-machine version of AT&T's System V for the 386 runs at about US$1500.

1.2.1 Where Linux comes in

Linux is a freely distributable version of UNIX developed primarily by Linus Torvalds [1] at the University of Helsinki in Finland. Linux was developed with the help of many UNIX programmers and wizards across the Internet, allowing anyone with enough know-how and gumption to hack a custom UNIX kernel the ability to develop and change the system. The Linux kernel uses no code from AT&T or any other proprietary source, and much of the software available for Linux is developed by the GNU project at the Free Software Foundation in Cambridge, Massachusetts. However, programmers all over the world have contributed to the growing pool of Linux software.

Linux supports almost all of the features of commercial versions of UNIX, as well as many not found on proprietary UNIX systems. Furthermore, Linux is closely compatible with the IEEE POSIX.1 standard, and has been developed with software portability in mind, thus supporting many important features of other UNIX standards. Linux also utilizes all of your system's memory—without memory limits or segmentation. The Linux system runs exclusively in 80386's protected mode, which allows it to operate as a true 32-bit operating system. A little-known fact about the 80386 and 80486 chips is that, internally, these processors were designed to support multitasking systems such as UNIX. Unfortunately, even MS-DOS 6.0 has yet to exploit these features.

There is a rumor abound that UNIX is a large, unwieldy system, unapproachably hungry for resources such as disk space and memory. In Linux, the proof to dispell those myths was implemented.

[1] `torvalds@kruuna.helsinki.fi`.

Linux is small, fast, and flexible. It requires fewer resources than IBM's OS/2, and uses less space than many MS-DOS or Microsoft Windows systems including large applications (such as Microsoft Word or Lotus 1-2-3). Linux has built-in support for networking, multitasking, and other features which you'll see touted as "New Technology" in systems such as in Windows NT. In fact, UNIX (and now, Linux) has implemented this "New Technology" for well over 15 years. The proprietary PC software industry is still catching up.

When using Linux, keep in mind that it is truly "a hacker's operating system"—developed by and for UNIX hackers. This is a strong distinction between commercial versions of UNIX, which are designed for customers, not for hackers. Commercial versions of UNIX are expected to work "out of the box". This is not the case with Linux.

Because of this distinction, many newcomers get easily frustrated with Linux. The only problem is a lack of basic UNIX know-how. Setting up and running your own UNIX system is something which most UNIX users never get to do, even after years of experience. Linux has provided UNIX newcomers the unique opportunity to "do it yourself"—but nobody said it was going to be easy.

It takes a lot of time and effort to run your own UNIX system (and a certain knack for fixing problems). Even with standard Linux distributions, such as SLS, there are sometimes little quirks which need to be fixed by hand in order for everything to work correctly. If you have previous UNIX experience, it should be easy to find these problems. However, if you're new to UNIX, it would serve you well to read up on using and running a UNIX system before you dive in.

Put simply, Linux isn't for everyone. Many users get in "over their heads" when getting started with Linux. To keep your head above water, we strongly encourage you to find a good book on using UNIX, as well as a book on UNIX system administration. In this manual, we include an introductory UNIX and system administration tutorial, but this is the bare minimum that is needed to get started.

The Linux Documentation Project is working to solve this dilemma. In the future, the LDP's *Linux User's Guide* and *Linux System Administration Guide* will be available. Until then, you'll need to rely on other sources.

See Appendix B for a list of relevant UNIX books and other sources of information.

1.3 About Linux's Copyright

Linux is copyrighted under the GNU General Public License, sometimes called the *GPL* or *copyleft*. This license was developed by the Free Software Foundation to allow programmers to write "free software", where "free" refers to freedom, not just cost. The GPL provides for the protection of such free software in a number of ways. First of all, it allows the original author to retain the software's copyright. Secondly, it allows others to take the software and modify it, or base other programs on it. Thirdly, it allows others to redistribute or resell the software, or modified versions of the software. You can even resell the software for profit. However, in reselling or redistributing the software, you cannot restrict any of these rights from the party you're selling it to. Also, if you sell the software, you have to provide at no cost the full source code so that others can modify the software and resell it if they wish.

This might sound a bit silly at first. Why sell software for profit, if someone else can give it away for free? Essentially, that's the point. However, let's say someone bundles a large amount of free software together on a CD-ROM and wishes to distribute it worldwide. They'll probably need to charge for this service to cover their cost and risk in distributing the software. Right now, several organizations are distributing Linux on CD-ROM, and making profit from it. The original authors of the Linux software may never see a penny of these revenues. This is allowed by the GNU GPL—the point of free software isn't to make money; nobody's getting ripped off. That's simply an understanding between the authors of the software and those who are using it or selling it.

One other thing: Free software, as covered by the GNU GPL (which includes Linux), comes with absolutely *no* warranty. However, individual vendors may provide support for the software, which usually includes a warranty. However, unless you purchased such support, the assumption is that the software comes with no such warranty, and if you use a piece of free software which goes haywire and wipes everything on your system, neither the authors nor those that distributed the software to you are liable[2].

Free software as covered by the GPL is not shareware, nor is it in the public domain. Neither of these terms correctly describe what free software really is. The complete GNU GPL is printed in Appendix E. To sum it all up, you can freely distribute Linux as much as you like, and you can even modify it and distribute your own version of Linux. But in doing so, you can't take away those rights from others. Furthermore, you must attribute the original authors of the work.

1.4 Features of Linux

Here are some of the benefits and features that Linux provides over single-user operating systems (such as MS-DOS) and other versions of UNIX for the PC.

- **Full multitasking and 32-bit support.** Linux, like all other versions of UNIX, is a real multitasking system, allowing multiple users to run many programs on the same system at once. The performance of a 50 MHz 486 system running Linux is comparable to many low- to medium-end workstations, such as those from Sun Microsystems and DEC, running proprietary versions of UNIX. Linux is also a full 32-bit operating system, utilizing the special protected-mode features of the Intel 80386 and 80486 processors.
- **GNU software support.** Linux supports a wide range of free software written by the GNU Project, including utilities such as the GNU C and C++ compiler, `gawk`, `groff`, and so on. Many of the essential system utilities used by Linux are GNU software.
- **The X Window System.** The X Window System is the de facto industry standard graphics system for UNIX machines. A complete version of The X Window System (known as "XFree86") is available for Linux. The X Window System is a very powerful graphics interface, supporting many applications. For example, you can have multiple login sessions in different windows on the screen at once. Other examples of X Windows applications are Seyon, a powerful telecommunications program; Ghostscript, a PostScript language processor;

[2] Hopefully, this won't happen.

and XTetris, an X Windows version of the popular game. See Section 6.1 for an introduction to the X Window System.

- **TCP/IP networking support.** TCP/IP ("Transmission Control Protocol/Internet Protocol") is the set of protocols which links millions of university and business computers into a worldwide network known as the Internet. With an Ethernet connection, you can have access to the Internet or to a local area network from your Linux system. Or, using SLIP ("Serial Line Internet Protocol"), you can access the Internet over the phone lines with a modem. Section 6.2 has an introduction to configuring the Linux TCP/IP software.
- **Virtual memory and shared libraries.** Linux can use a portion of your hard drive as virtual memory, expanding your total amount of available RAM. Linux also implements shared libraries, allowing programs which use standard subroutines to find the code for these subroutines in the libraries at runtime. This saves a large amount of space on your system, as each application doesn't store its own copy of these common routines.

1.5 Hardware Requirements

Unlike some other versions of UNIX for the PC, Linux is very small. You can run an entire system from a single high-density 5.25" floppy. However, to run a complete Linux system, there are other hardware requirements, listed below.

Linux, by its very nature, is continuously expanding, and more features are added every day. However, hardware compatibility is limited to that hardware which the developers themselves have access to. For instance, if none of the Linux developers has access to the Foobaz Net-O-Driver 3000 card, then chances are it isn't supported. On the other hand, there are many "generic" drivers—for example, the IDE disk driver for Linux should work with all IDE hard drives and adapters.

If your favorite peripheral isn't supported by Linux, all that's required is to write a kernel driver for it. This may be easy or difficult, depending on the hardware and the technical specifications that are available. For example, some hardware developers prefer to write their own drivers for MS-DOS and Windows, and not release specifications for third parties to write their own. Therefore, writing drivers for Linux will be difficult, if not impossible. (We call this practice "bad business" for the hardware vendors involved.)

1.5.1 Suggested setup

Below, the suggested hardware configuration for Linux is listed. If you're in the market for a new system, you should follow the recommendations given below. In the next section, we'll talk about some of the hardware drivers which are currently under development for Linux, and other hardware which is known not to work.

- An Intel 80386 or 80486 based system. You don't need a math coprocessor, although it's strongly recommended that you have one. (If you have an 80386 chip, 80387 math coprocessors are available separately, and are installed in a socket on your motherboard. If you have an

80486 processor, the math coprocessor is on the 486 chip itself. The exception is the 80486SX, which is a 486 chip without the coprocessor components.) If you don't have a math coprocessor, the Linux kernel will emulate floating-point math for you. If you do have one, however, floating point math will be handled by the hardware, which for some applications is a real plus.

The 386SX, 486SX, and the accelerated 486DX and 486DX2 chips are all reported to work. It is suggested that you have a genuine Intel processor, instead of a clone (such as a CTX processor). Look for the "Intel Inside" logo on the machine. The forthcoming Intel Pentium chip should work with Linux without any changes.

- Your system must be either an ISA, EISA, or local bus architecture machine. These terms specify how the CPU communicates with hardware, and are a characteristic of your motherboard. Most existing systems use the ISA bus architecture. The EISA bus is a newer form of the ISA bus, which is reportedly faster on some machines. Local bus architecture is often the fastest of the three, as it allows the CPU to communicate directly to video and drive adapters. If your machine uses local bus, it is strongly recommended that it comply with the VESA Local Bus standard (most local bus systems do).

  MicroChannel architecture (MCA) machines, such as the IBM PS/2 line, are not currently supported.

- At least four megabytes of RAM. Technically, Linux is capable of running on a system with only two megabytes, however, some distributions of Linux require four megabytes for installation. Memory is speed, so if you have more physical RAM you'll thank yourself for it later. If you're a "power user," 8 megabytes should be more than enough for most applications.

- An AT-standard compatible hard drive controller. This includes MFM, RLL, ESDI, and IDE controllers. Many SCSI controllers are also supported. These terms specify the means used to communicate with the hard drive through the controller card; most controllers are either IDE or SCSI.

  Drivers for XT-standard driver controllers are under development; see the next section.

- A hard drive (compatible with your controller card, of course), with free space available for installing Linux. The amount of space required depends on the amount of software you're installing and how much free space you wish to leave yourself. If you only install a small amount of software, less than 10 megabytes is required. However, if you install a number of optional software packages, including the X Window System, perhaps 80 megabytes or more, including space for users, would be required. In addition, you will probably want to set aside some amount of space on your drive as a swap partition, used for virtual memory. This is covered in Chapter 3.

- A Hercules, CGA, EGA, VGA, or Super VGA video card and monitor. In general, if your video card and monitor work under MS-DOS or Microsoft Windows, then Linux should be able to use them without any problem. However, if you're going to use the X Window System, there are certain hardware configurations which are not yet supported. See Section 6.1 for more.

1.5.2 Other hardware

There are other hardware drivers currently under development for Linux. However, to use these drivers, you usually have to patch them into your kernel code, which assumes that you already have a running Linux system.

The Linux FAQ has information on patching in hardware drivers with Linux. See Appendix B for more information.

Linux will also run on a number of laptop machines (some laptops use certain software interrupts to power the memory, and Linux doesn't cooperate with these systems, yet). The best way to find out if Linux will run on your hardware is just to try it out.

1.6 Before You Get Started

Assuming that you have hardware compatible with Linux, obtaining and installing the system is not difficult. However, there are a number of burning questions which most users want answers to before they dive in.

1.6.1 Running other operating systems with Linux

You can install other operating systems, such as MS-DOS or OS/2, along with Linux on the same machine. Each operating system uses its own partitions on the hard drive, and they do not interfere with each other. For example, if you want to install both MS-DOS and Linux on the same drive, fine. However, MS-DOS will use one partition on the drive, and Linux will use another. Partitioning your drive for use with Linux is covered in Section 3.2.3.

To select which operating system to run at boot time, you can install LILO, a program which replaces the standard MS-DOS boot loader on your hard drive. With LILO installed, at boot time you select the operating system to run from a simple menu. Or, you can set a default operating system to boot, and override the default by holding down the shift key as the system is booting. LILO is covered in detail in Section 5.2.2.

Alternately, if you use OS/2, you can boot Linux from the OS/2 Boot Manager. See Section 5.2.2 for more information.

1.6.2 Accessing MS-DOS files from Linux

Linux supports several features which will allow you to access your MS-DOS files from Linux. The `mtools` package, included with most distributions of Linux, allows you to use commands such as `mcopy` and `mdir` to access your MS-DOS files. Or, you can mount an MS-DOS partition or floppy directly under Linux, giving you direct access to your files using the MS-DOS filesystem. (See Section 6.1.6.)

There is also an MS-DOS Emulator available for Linux, and work is beginning on a Microsoft Windows emulator to run under the X Window System. The MS-DOS Emulator isn't perfect; don't

expect to play Wing Commander from Linux. However, it will enable you to run many standard MS-DOS applications, such as Wordperfect or Lotus 1-2-3. Watch out, Microsoft!

1.6.3 How to pronounce *Linux*

Pronouncing the word "*Linux*" is one of the great mysteries of the Linux world. Americans pronounce the name *Linus* with a long *i* sound, as in *pie*. However, because Linux was originally based on a small, PC-based implementation of UNIX called "Minix" (pronounced with a short *i*), the actual pronunciation of *Linux* preserves this characteristic—it's *LIH-nucks*. Think Finnish.

1.7 Finding Help

There are many people out there who enjoy helping new users get started with Linux. If you have any problems with installing or using the system, the first thing you should do is consult this manual (and the other Linux manuals) to see if your problem is covered there. Another good source of information is the Linux Frequently Asked Questions list, which covers many disparate topics not in this manual. Appendix B for information on the Linux FAQ and other sources of information.

If you have access to USENET news, you can read and post to the newsgroups `comp.os.linux.help`, `comp.os.linux.development`, `comp.os.linux.admin`, and `comp.os.linux.misc`[3]. The moderated newsgroup `comp.os.linux.announce` is also available for announcements and important information. However, it's strongly suggested that you read all of the available documentation before posting questions; most problems are covered in this manual and the Linux FAQ list.

You can also direct questions about this manual, or Linux in general, to the authors of this book. We're very open to any questions, comments, or suggestions. Matt Welsh can be reached on the Internet at `mdw@sunsite.unc.edu` and Lars Wirzenius at `wirzeniu@cc.helsinki.fi`.

[3] These newsgroups are to be created on 11 August 1993, and supercede `comp.os.linux`. If you do not have access to these four groups, encourage your news administrator to create them. `comp.os.linux` should no longer be used.

Chapter 2

Getting Linux

This chapter covers how to obtain the Linux software. If you received this manual with a set of Linux software, on disk or CD-ROM, then you can go ahead and skip to Chapter 3 on installing the system.

2.1 Distributions of Linux

There is no single, "official" release of the Linux software. In fact, there are many releases, each independently coordinated by various companies and individuals. While these releases differ somewhat in their goals and specifications, each release has a number of elements in common with the rest:

- The Linux **kernel**—the lowest level program of the Linux system, the operating system itself. The kernel is constantly running, handling the low-level details of memory management, process scheduling, device access, and so on. In any UNIX system, the kernel is the software which controls access to system resources (such as memory and processor time) from user programs (such as a text editor). This is in sharp contrast to operating systems such as MS-DOS, which allow only a single program to run at a time.
- A set of utilities which handle basic system tasks such as manipulating files, creating users, reporting system activity, and so on. These programs are essentially the same on any UNIX system, and provide a standard interface for users. For example, the command to copy a file on all UNIX systems is known as `cp`. Therefore, using Linux will be very similar, if not identical, to using other versions of UNIX.
- The system **libraries**, which contain common subroutines used by software to handle everything from file I/O to displaying graphics. Linux implements **shared libraries**, which allow programs to share the code in these subroutines, reducing the size of program binaries significantly.

- A set of system configuration files, containing information about the users on the system, available devices, network interface configuration, and so on. Many individual applications also have their own set of configuration files.
- Online documentation, in the form of manual pages (often abbreviated as "man pages"). Man pages document the commands, library routines, kernel system calls, file formats, and device driver interfaces available on the system. They are an excellent source of online help and are easy to use. The man pages available for Linux are incomplete, but most of the important system commands and library calls are documented.
- And, of course, application software. There are many applications available for Linux, such as TEX (a document processing system), Emacs (a powerful yet baroque text editor), the X Window System (a graphics interface), and more.

The application software is what really composes the Linux system from the user's point of view. However, it is all of these elements combined which form the entire system. As mentioned before, there isn't a single official distribution for all of this software. There are many distributions available—each of which differ somewhat in their installation methods and supported software, but all of which contain the essential elements listed above.

In this book, we focus on the Softlanding Linux System (SLS) distribution of Linux. SLS has become the de facto standard for Linux installations worldwide. Because of its completeness and ease of use, we will only cover how to get and install the SLS distribution here.

We don't mean to be biased towards a particular release; however, it takes a lot of time and effort to keep up with all of the Linux distributions available. However, all of the concepts discussed in this book apply to Linux distributions other than SLS.

2.2 The SLS Distribution

The Linux distribution that many people have installed is known as the Softlanding Linux System ("SLS") distribution. SLS is coordinated by Peter MacDonald[1], and has become the most widely-used release of Linux, because of its completeness and ease of installation.

The SLS distribution is broken into various sets of disks, each of which is denoted by a letter followed by a disk number. For example, the `a` series disks are numbered `a1`, `a2`, `a3`, and `a4`. The disk series in the current SLS distribution are shown below.

- `a1-a4`: The minimal base system, required. Contains the Linux kernel, libraries, basic binaries, and tools used to set up and install the system.
- `b1-b7`: Base system extras, such as man pages, Emacs, and so on.
- `c1-c3`: Compilers, such as gcc, p2c, f2c, and others.
- `x1-x10`: The X Window System and related software.

[1] `pmacdona@sanjuan.uvic.ca`.

- `t1-t3`: TEX, a document formatting system.
- `s1`: Source code for several of the packages. Currently under development.
- `d1-d2`: Documentation for various things.

As more packages are added to SLS, new series are added, or extra disks are added to the existing series.

The only set of disks which you must install are the `a` set. However, you should probably install at least the `b` and `c` disk sets as well, which contain the full base system and compilers. For X Windows (see Section 6.1) you should install the `x` series in addition to `a`, `b`, and `c`.

It is easy to upgrade your software with the SLS distribution. For example, you could install only the `a` and `b` series, and install the `c` and `x` series at another time. Therefore, you may wish to only install the SLS `a` series at first, to try out a minimal system, and install the rest later.

The amount of disk space required for installing the SLS release depends on how many of the series you install. Table 2.1 summarizes the disk space requirements for installing each SLS package.

Package	Disk series	Approximate space requirements
Tiny	a	15 megabytes
Base	a, b, c	45 megabytes
Main	a, b, c, x	70 megabytes
Full	a, b, c, x, d, s, t	90 megabytes

Table 2.1: SLS Disk Space Requirements

Also keep in mind that in addition to the space requirements listed in Table 2.1, you'll need to set aside extra space for swap (see Section 3.2.3) and for user accounts, free space to install new software, and so on.

2.2.1 Getting SLS from the Internet

If you have access to the Internet, you can download the SLS distribution via FTP from a number of sites. If you've never used FTP before, see Appendix C for a quick tutorial. If you don't have Internet access, you can get SLS either via mail or from a number of BBS's worldwide; see sections 2.2.2 and 2.3 for more information.

2.2.1.1 Downloading the files

The two primary FTP sites from which to obtain the SLS distribution are `tsx-11.mit.edu` and `sunsite.unc.edu`. In addition, a large number of FTP sites around the world mirror these two—see Appendix C for a listing of Linux FTP sites.

The SLS release is kept in the directory `/pub/linux/packages/SLS` on `tsx-11.mit.edu`, and in `/pub/Linux/SLS` on `sunsite.unc.edu` (note that the filenames are case-sensitive).

You should download the following files. Be sure to use binary mode when downloading them.

- If you boot from a 3.5" floppy, download the file `a1.3`. If you boot from a 5.25" floppy, download `a1.5` instead. These files are images of the SLS `a1` disk which is used for booting the system[2].

  Hereafter, we'll be referring to the files `a1.3` and `a1.5` simply as "`a1`"—because you'll only be downloading one or the other.

- `rawrite.exe`: A DOS program to "raw write" a given file, block by block, to an MS-DOS formatted floppy. You will use `rawrite` to create the `a1` disk.

  Note that `rawrite.exe` may not be in the SLS directory, but it should be somewhere nearby on the FTP site. On `sunsite.unc.edu`, `rawrite` is found in:

  ```
  /pub/Linux/SLS/DOSUTILS/rawrite2.exe
  ```

  On `tsx-11.mit.edu`, `rawrite` is found in:

  ```
  /pub/Linux/INSTALL/dos_utils/rawrite.exe
  ```

- The files in the `a2`, `a3` and `a4` directories. Be sure to keep these files in separate directories when you download them. Later, you'll simply copy these files to MS-DOS floppies, but you don't want to get the `a2`, `a3` and `a4` files mixed up.

- If you plan to get any of the other series, you need to download the files in their corresponding directories. For example, if you're downloading the `b` disk series, grab the files in the directories `b1` through `b7`. As with `a2` through `a4`, keep the files in each directory separate as you download them.

- The files `SLS.FAQ` and `SLS.README`, which are text files giving important information about the current release of SLS.

2.2.1.2 Transferring the files to MS-DOS

Once you have these files, transfer all of them to an MS-DOS system, in order to create the installation disks[3].

If you did not FTP the files directly to an MS-DOS system, you may need to ask your local computer guru for information on transfering the files to MS-DOS. The methods used to do this vary from site to site.

[2] Some FTP sites have the files `a1.3.Z` and `a1.5.Z` instead. These are compressed images of the `a1` disk; you'll need to use the UNIX `uncompress` command to uncompress it. We've requested that FTP sites leave the files `a1.3` and `a1.5` uncompressed for this step to be unnecessary for those downloading to a non-UNIX machine; both `tsx-11` and `sunsite` should have them both in uncompressed format.

[3] If you have access to a UNIX machine with a floppy drive, you can create the installation disks from there. See Section A.4.

2.2.1.3 Making the installation disks

To create the installation disks, you'll be using `rawrite.exe` as well as the MS-DOS `COPY` command. First, be sure that you have one MS-DOS formatted, blank floppy for each disk in the SLS series that you downloaded[4]. All of the floppies must be formatted high density (either 1.2Mb 5.25" or 1.44Mb 3.5").

The floppy used for the `a1` disk must be the type of floppy that you boot from on your system. To create the `a1` disk, run the command

```
C:\> rawrite
```

from MS-DOS. You should see something like

```
RaWrite 1.2 - Write disk file to raw floppy diskette
Enter source file name:
```

Specify the filename that you wish to rawrite (such as `a1.3` or `a1.5`), followed by the drive that you wish to write it to (either `A:` or `B:`).

```
Enter source file name:  a1.5
Enter destination drive:  A:
Please insert a formatted diskette into drive A: and press -ENTER- :
```

Press return and `rawrite` will copy the file to the disk in the specified drive.

```
Number of sectors per track for this disk is 15
Writing image to drive A:.  Press ^C to abort.
Track:  00 Head:  0 Sector:  1
...
Done.
```

Once `rawrite` is done, the floppy will no longer be accessible by MS-DOS. The floppy has been completely overwritten with the `a1` image, and is now a "Linux-format" floppy.

If you have problems with `rawrite`, make sure that the floppy is MS-DOS formatted, high density. If the floppy has any bad sectors you may just need to use another disk.

The rest of the disks are created using the MS-DOS `copy` command. All of these disks must be of the same type (either 3.5" or 5.25") and must be high-density MS-DOS formatted as well. Copy all of the files from the `a2` directory to a floppy labeled `a2`. For example,

For example,

```
C:\A2> copy *.* A:
```

[4] It is possible to install SLS directly from your hard drive, instead of from floppies. See Section A.1 for more information.

will copy all of the files in the directory `C:\A2` to the disk in `A:`.

Repeat the procedure for the rest of the disks in the SLS distribution. Remember that you must have at least `a1` through `a4`. After you are finished creating the disks, you can install the software. This is covered in Chapter 3.

2.2.2 Getting SLS via U.S. Mail

The SLS distribution is also available on floppy via U.S. Air or Land mail. A small fee is charged on a per-disk basis, but this fee is nominal and more than worth the service. This is a reliable way to get the SLS distribution, even if you do have network access—it saves you a lot of time downloading and creating the disks yourself.

To order, send a check or money order to:

Softlanding Software
910 Lodge Ave.
Victoria, B.C., Canada
V8X-3A8

By the beginning of 1993, they hope to take Visa and/or Mastercard orders by telephone. Their number is +1 604 360 0188. They do take international orders, so if you're not in the States or Canada you can still get the SLS distribution by mail from this address.

The charge is on a per disk basis. The complete distribution (that is, all of the `a`, `b`, `c`, `x`, and `t` series) is currently 29 disks, but more disks are added to the distribution on occasion.

The copying charge is $3.25/disk US ($4.00/disk Canadian). For 3.5" disks, add $1.00/disk. There is also a $10.00 shipping and handling charge. Table 2.2 summarizes the various packages and prices.

Package	#Disks	Series	5.25" Disks	3.5" Disks
Tiny	4	a	US $23.00 (CDN $26.00)	US $27.00 (CDN $30.00)
Base	16	a,b,c	US $62.25 (CDN $74.00)	US $78.00 (CDN $90.00)
Main	24	a,b,c,x	US $88.00 (CDN $106.00)	US $112.00 (CDN $130.00)
Full	29	a,b,c,x,t	US $104.25 (CDN $126.00)	US $133.25 (CDN $155.00)

Table 2.2: SLS by-mail disk prices

2.3 Getting Linux from BBSs

Using a modem and standard communications software, you can also download Linux from a number of bulliten board systems (BBSs) worldwide. You'll likely find a version of the SLS distribution on these BBS sites, with each disk image packed into a PKZIP `.zip` archive file. The PKUNZIP utility,

used to unpack these files before you transfer the files to floppies, should be easy to locate (perhaps on the BBS itself).

For the most part, you can follow the directions for creating the SLS floppies in Section 2.2.1.3 once you have downloaded the files. Just be sure that the files on the `a2`, `a3`, etc. disks are expanded—in other words, the SLS `a2` contains a number of `.tgz` files. If you download a single `a2.zip` from the BBS, you may need to use PKUNZIP to get those `.tgz` files before you place them on the floppy.

In any case, there should be a `README` describing how to load and install the release of Linux found on the BBS. What version and distribution of Linux you find on these BBS's is really up in the air—we can make no guarantees that you'll find a particular release in a particular format, or that the files on the BBS are up-to-date. Every BBS is different.

If you're unfamiliar with downloading files from BBS's, it's really quite simple. You need a modem and communications software, to begin with. The communications software should come with complete instructions on how to dialup a BBS and how to download software—the mechanism differs greatly depending on your modem and your software. If all else fails, have a computer guru near you show you how it's done.

Printed in Appendix D is a list of BBS sites which carry Linux software. You should be able to find a BBS within reasonable calling distance. If not, and if you don't have Internet access of some kind, you may want to consider ordering the SLS distribution of Linux via mail. See Section 2.2.2 for details.

Chapter 3

Installing Linux

In this chapter, we'll go through the steps of installing Linux, from repartitioning your drive, to setting up the Linux filesystems, to actually installing the software.

As you're partitioning your drive and installing Linux, it's always a good idea to take notes of every command you type and what goes on during the process, just so you have something to refer to when you run into those inevitable problems. Trust me; it's worth the time it takes just to jot down notes while installing, especially if you're a newcomer to Linux. If you have problems later and have to ask for help, having a list of *exactly* what commands you typed and what the results were will be invaluable.

3.1 SLS installation overview

This is an overview of installing the SLS release of Linux. As mentioned before, the basic ideas are applicable to distributions other than SLS as well.

To install the SLS release, you're doing to do the following:

- Backup any software (for MS-DOS, OS/2, etc.) on your system. (Section 3.2.3.)
- Repartition your drive to allocate space for Linux. (Section 3.2.3.)
- Reinstall the original software on your system. (Section 3.2.3.)
- Boot the SLS `a1` disk, and create partitions for Linux. (Section 3.3.4.)
- Create **filesystems** on those partitions. (Section 3.3.7.)
- Finally, install the software. (Section 3.4.)

3.2 Repartitioning your Drive

As we have mentioned, in order to allocate space for Linux on your hard drive(s), you'll need to *repartition* the drive(s), resizing any existing partitions.

Until now, most users of 386 and 486 systems haven't had to worry about partitioning their hard drive, or even running more than one operating system on the same drive (unless you're an OS/2 user). Thus, some explanation of hard drives and partitions is probably in order.

3.2.1 Hard drive geometry

A typical hard drive consists physically of one or more circular **platters** which rotate about a central axis. The read/write **heads** move to specified place on the disk surface to access information. There is typically one head on each side of every platter, which all move in unison back and forth across the platter (that is, either closer to the center of the disk, or closer to the outer edge).

The drive platters are divided into **cylinders**, which is the area of each platter which can be accessed without moving the heads. A cylinder is a barrel-shaped cross section of a disk, consisting of a circular strip from each side of each platter. The part of a cylinder which is the circular strip on a single platter is called a **track**.

Thus, if there are 3 platters in a given drive (stacked on each other, like a sandwich), then you have a read/write head on each side of each platter, to make up 6 heads. (In actuality, the outer surface of the outermost platters don't usually have heads accessing them, so in this case you'll have only 4 heads.)

Thus if you position the heads at a single radius from the center of the disk, the area which they can access by rotating the platters is one cylinder. A track is the area which one head can access. Thus, for our drive with 4 heads, there are 4 tracks per cylinder.

Each track is divided into a number of pie-shaped sliced called **sectors**, which are the smallest parts of the disk which can be read or written at one time.

Therefore, the **geometry** of the disk is specified as

- The number of cylinders that the disk contains;
- The number of tracks per cylinder (same as the number of heads);
- The number of sectors per track; and,
- The size of each sector (in bytes).

Some of these numbers will vary, but a typical disk might have about 1000 cylinders, 6 heads, 15 or 20 sectors per track, and each sector containing 512 bytes. The size of the disk is

(1000 cylinders) * (6 tracks per cylinder) * (15 sectors per track) * (512 bytes per sector)

or 46,080,000 bytes (about 46 megs).

3.2.2 Partitions

Hard drives are divided into one or more **partitions** which are sections of the hard drive set aside for some operating system to use. The concept of a partition allows you to have, for instance, MS-DOS running on one partition on your hard drive, and Linux on another.

Many MS-DOS systems have only a single partition taking up the entire drive. To MS-DOS, this partition is known as `C:`. If you have more than one partition on your drive, MS-DOS names them `D:`, `E:`, and so on. In a way, each partition acts like a separate hard drive.

On the the first cylinder, first head, and first sector of the disk is a 1-sector **master boot record** along with a **partition table**. The boot record (as the name implies) is used to boot the system. The partition table contains information about the locations and sizes of your partitions.

There are three kinds of partitions: **primary**, **extended**, and **logical**. Of these primary partitions are by far used most often. However, because of a limit in the size of the partition table, you can only have 4 primary partitions on any given drive.

The way around this 4 partition limit is to use an extended partition. An extended partition doesn't hold any data by itself; instead, it acts as a "container" for logical partitions. Therefore, you could create one extended partition, covering the entire drive, and within it create several logical partitions. However, you may have only one extended partition per drive.

3.2.3 Resizing your current partitions

If your hard drive is currently taken up by partitions for other operating systems, you'll need to delete and resize those partitions to allocate space for Linux. If you don't have any other operating systems installed (you're installing Linux on a clean system), you can skip to Section 3.3.

There isn't any safe or reliable way to resize an existing partition without deleting it[1]. Therefore, before repartitioning your drive, **backup your system**. After you have repartitioned the drive, you can reinstall your original software from the backups. Also keep in mind that because you'll be shrinking your original partitions, you may not have space to reinstall everything.

How much space should you leave free for Linux? This depends greatly on how much software you're installing, and how much free space you want to leave yourself. In Section 3.3.2 we'll cover this in detail.

Under MS-DOS, the program used to modify partitions is `FDISK`[2] In order to repartition, you should create an MS-DOS bootable "system disk" which contains all of the necessary DOS utilities, including `FDISK.EXE` and `FORMAT.COM`[3]. Boot from this disk, and run `FDISK`.

Use of `FDISK` should be self-explanatory, but consult your MS-DOS documentation for details. When you start `FDISK`, use the menu option to display the partition table, and *write down* the

[1] There are several programs available for MS-DOS which are able to resize partitions nondestructively. One of these is known as "`FIPS`", and can be found on many Linux FTP sites.

[2] Instructions for modifying partitions for operating systems other than MS-DOS are similar to those presented here. Refer to the documentation for those systems, if applicable.

[3] You can format an MS-DOS system disk with the command `format /s A:`.

information displayed there. It is important to keep a record of your original setup in case you want to back out of the Linux installation.

To delete an existing partition, choose the `fdisk` menu option to `Delete an MS-DOS Partition or Logical DOS Drive`. Specify the type of partition that you wish to delete (primary, extended, or logical) and the number of the partition. Verify all of the warnings. Poof!

To create a new (smaller) partition for MS-DOS, just choose the `fdisk` option `Create an MS-DOS Partition or Logical DOS Drive`. Specify the type of partition (primary, extended, or logical), and the size of the partition to create (specified in megabytes). `fdisk` should create the partition and you're ready to roll.

After you're done using `fdisk`, you should exit the program and reformat the new partition(s). For example, if you resized the first DOS partition on your drive (`C:`) you should run the command

```
format /s C:
```

You may now reinstall your original software from backup.

◇ **A warning:** When repartitioning, you should only use the version of `fdisk` for the operating system that you are manipulating partitions for. That is, you should not use MS-DOS's `fdisk` to make partitions for Linux, and vice versa. In some cases, doing so won't cause any damage, but will cause MS-DOS not to boot correctly. For manipulating MS-DOS partitions, use MS-DOS's version of `fdisk`, and for Linux partitions use Linux's version of `fdisk`.

3.3 Creating your Linux Partitions and Filesystems

Setting up partitions for Linux is very similar to setting them up for other operating systems such as MS-DOS. To create your Linux partitions, you use the Linux version of the `fdisk` program. After you create the partitions, you create **filesystems** on them, which allows them to be used by the Linux system. Making a filesystem is analogous to "formatting" a partition under MS-DOS.

Because your swap partition doesn't hold files, we call preparing it "making the swap space".

3.3.1 Drives and partitions under Linux

Drives and partitions under Linux are given different names than their MS-DOS counterparts. Under MS-DOS, floppy drives are referred to as `A:` and `B:`, while hard drive partitions are named `C:`, `D:`, and so on. Under Linux, the naming convention is quite different, but easier to comprehend.

Device drivers, found in the directory `/dev`, are used to communicate with devices on your system (such as hard drives, mice, and so on). For example, if you have a mouse on your system, access to it is through the driver `/dev/mouse`.

Floppy drives, hard drives, and individual partitions are all given device drivers through which you access them. Table 3.1 lists the names of these various device drivers.

Device	Name
First floppy (`A:`)	`/dev/fd0`
Second floppy (`B:`)	`/dev/fd1`
First hard drive (entire drive)	`/dev/hda`
First hard drive, primary partition 1	`/dev/hda1`
First hard drive, primary partition 2	`/dev/hda2`
First hard drive, primary partition 3	`/dev/hda3`
First hard drive, primary partition 4	`/dev/hda4`
First hard drive, logical partition 1	`/dev/hda5`
First hard drive, logical partition 2	`/dev/hda6`
⋮	
Second hard drive (entire drive)	`/dev/hdb`
Second hard drive, primary partition 1	`/dev/hdb1`
⋮	
First SCSI hard drive (entire drive)	`/dev/sda`
First SCSI hard drive, primary partition 1	`/dev/sda1`
⋮	
Second SCSI hard drive (entire drive)	`/dev/sdb`
Second SCSI hard drive, primary partition 1	`/dev/sdb1`
(and so on...)	

Table 3.1: Linux partition names

A few notes about this table. Note that `/dev/fd0` corresponds to the first floppy drive (`A:` under MS-DOS) and `/dev/fd1` corresponds to the second floppy (`B:`).

Also, SCSI hard drives are handled differently than other drives. IDE, MFM, and RLL drives are accessed through the devices `/dev/hda`, `/dev/hdb`, and so on. The individual partitions on the drive `/dev/hda` are `/dev/hda1`, `/dev/hda2`, and so on. However, SCSI drives are named `/dev/sda`, `/dev/sdb`, etc., with partition names such as `/dev/sda1` and `/dev/sda2`.

Here's an example. Let's say that you have a single IDE hard drive, with 3 primary partitions. The first two are set aside for MS-DOS, and the third is an extended partition which has two logical partitions, both for use by Linux. The devices referring to these partitions would be:

First MS-DOS partition (`C:`)	`/dev/hda1`
Second MS-DOS partition (`D:`)	`/dev/hda2`
Extended partition	`/dev/hda3`
First Linux logical partition	`/dev/hda5`
Second Linux logical partition	`/dev/hda6`

Note that `/dev/hda4` is skipped; it corresponds to the fourth primary partition, which we don't have. Logical partitions start with `/dev/hda5` and go on up.

3.3.2 Partition sizes for Linux

For Linux, you will need to create at least two partitions. One will be used as the **root partition**, which contains the Linux software[4]. The second will be used as the **swap partition**, which acts as virtual memory on your system[5]. Swap space is very important for Linux: it allows you to run many more programs at once than you would be able to otherwise. This is especially important if you're low on physical RAM, or if you're running X Windows. Even if you have 16 megabytes or more of physical RAM, using swap space on your hard drive is a good idea.

The size of your root partition depends greatly on how much software you're installing. Section 2.2 should give you an idea of the diskspace requirements for the SLS distribution of Linux. For example, a minimal SLS installation (only the `a` series disks) requires about 15 megabytes for the root partition, while a full installation requires 90 megabytes. You should also leave yourself room for expansion—space to install new software, space for users, and so on.

The size of your swap partition depends on the amount of physical RAM you have, and how much swap space you want. Swap space is used as virtual memory by Linux. Remember that Linux is a multitasking operating system; this allows you to run many programs at once. Because the number of simultaneously running programs may require more memory than you have, use of swap space is vital. Also, the more swap space you have, the less Linux will need to swap programs back and forth between real memory and virtual memory.

It is generally suggested that you have twice the amount of swap space than you have physical RAM. For example, if you have 4 megabytes of physical RAM, an 8-megabyte swap partition should suffice. However, the decision is really up to you. If you have 16 megabytes or more of physical memory, an 8- or 16-megabyte swap partition should be more than enough. However, your swap partition cannot be larger than 16 megabytes. You can use multiple swap partitions (up to 8) if you wish—if you'd like to have 32 megabytes of swap, simply create two 16-megabyte swap partitions.

3.3.3 Booting SLS

You are now ready to boot the SLS release on your system to start the installation.

To boot SLS, simply boot the `a1` disk on your system. While booting, hold down the ctrl or alt key. You should see the prompt

```
LILO
```

Followed by a boot menu.

At this point, you have two options: to boot the floppy with the ramdisk enabled, or to boot without the ramdisk. If you have at least 4 megs of RAM in your machine, then you can just press return or type

[4] You may choose to create separate partitions for each part of your Linux directory tree, using multiple filesystems. If you have UNIX system administration experience, you may be able to partition you drive more creatively. See Section A.3 for information.

[5] Instead of using a swap partition, you may use a swap *file*. See Section A.5 for more information.

```
ramdisk
```

to boot with the ramdisk enabled. This will speed up operations while you're using the `a1` disk. However, if you have only 2 megs of RAM in your machine, you'll need to type

```
floppy
```

to boot without the ramdisk.

The other options on the menu are for installing the Mitsumi CD-ROM release of SLS (see Section A.2) or for booting Linux from the hard drive (after you've installed it).

As the system boots, you'll be asked to

```
Press <return> to see SVGA-modes available or <space> to continue
```

Press space to use the standard 80x25 mode (not all SVGA modes work on all displays). There will be a short pause wile the system uncompresses the kernel (you should see "`Uncompressing Linux`" at the top of your screen. Also, there may be a 5-10 second pause after the `lp_init`... line at bootup, while the system checks for SCSI devices. Don't panic! Finally, you should see the message

```
Welcome to the SLS install/bootdisk.
```

and be given the login prompt

```
softland login:
```

Here, login as `root` (no password, of course).

```
softland login:  root

softland:/#
```

To look at the installation information on the disk, use the command

```
softland:/# install.info
```

Instead, if you login as `install`, you will be presented with an installation menu. In the following sections, we cover how to install the system by hand, instead of using the SLS menus. This is because manual installation is applicable to Linux distributions other than SLS, and also because it's a good idea to know how to use the commands directly (without the menu). However, the concepts behind using the SLS install menu and using the commands directly are the same.

3.3.4 Running `fdisk`

After booting Linux, run `fdisk` by typing

```
fdisk ⟨drive⟩
```

where ⟨*drive*⟩ is the Linux device name of the drive you plan to add partitions to (see Table 3.1 on page 20). For instance, if you want to run **fdisk** on the first SCSI disk in your system, use the command **fdisk /dev/sda**. **/dev/hda** (the first IDE drive) is the default if you don't specify one.

If you are creating Linux partitions on two separate drives, run **fdisk** once for each drive.

```
# fdisk /dev/hda
Command (m for help):
```

Here **fdisk** is waiting for a command; you can type **m** to get a list of options.

```
Command (m for help): m
Command action
a toggle a bootable flag
d delete a partition
l list known partition types
m print this menu
n add a new partition
p print the partition table
q quit without saving changes
t change a partition's system id
u change display/entry units
v verify the partition table
w write table to disk and exit
x extra functionality (experts only)
Command (m for help):
```

The **n** command is used to create a new partition. Most of the other options you won't need to worry about. To quit **fdisk** without saving any changes, use the **q** command. To quit **fdisk** and write the changes to the partition table to disk, use the **w** command.

The first thing you should do is display your current partition table and write the information down, for later reference. Use the **p** command.

```
Command (m for help): p
Disk /dev/hda: 16 heads, 38 sectors, 683 cylinders
Units = cylinders of 608 * 512 bytes

   Device Boot  Begin   Start     End  Blocks   Id  System
/dev/hda1   *       1       1     203   61693    6  DOS 16-bit >=32M


Command (m for help):
```

In this example, we have a single DOS partition on `/dev/hda1`, which is 61693 blocks (about 60 megs). This partition starts at cylinder number 1, and goes to cylinder 203. We have a total of 683 cylinders in this disk; so there are 480 cylinders left to make Linux partitions on.

To create a new partition, use the `n` command. In this example, we'll create two primary partitions (`/dev/hda2` and `/dev/hda3`, respectively) for Linux.

```
Command (m for help): n
Command action
e extended
p primary partition (1-4)
```

Here, `fdisk` is asking the type of the partition to create: extended or primary. In our example, we're creating only primary partitions, so we choose `p`.

```
Partition number (1-4):
```

`Fdisk` will then ask for the partition to create; since partition 1 is already used, our first Linux partition will be number 2.

```
Partition number (1-4): 2
First cylinder (204-683):
```

Now enter the starting cylinder number of the partition. Since cylinders 204 through 683 are unused, we'll use the first available one (number 204). There's no reason to leave empty space in between your partitions.

```
First cylinder (204-683): 204
Last cylinder or +size or +sizeM or +sizeK (204-683):
```

`Fdisk` is asking for the size of the partition to create. We can either specify an ending cylinder number, or a size in bytes, kilobytes, or megabytes. Since we want our partition to be 80 megs in size, we specify `+80M`.

```
Last cylinder or +size or +sizeM or +sizeK (204-683): +80M
Warning: Linux cannot currently use 33090 sectors of this partition
```

This warning can be ignored. `Fdisk` prints the warning because it's an older program, and dates before the time that Linux partitions were able to be larger than 64 megabytes.

Now we're ready to create our second Linux partition. For sake of demonstration, we'll create it with a size of 10 megabytes.

```
Command (m for help): n
Command action
e extended
```

```
p primary partition (1-4)
p
Partition number (1-4): 3
First cylinder (474-683): 474
Last cylinder or +size or +sizeM or +sizeK (474-683): +10M
```

Lastly, we'll display the partition table. Again, write down all of this information—especially the block sizes of your new partitions. You'll need to know the sizes of the partitions when creating filesystems, later.

```
Command (m for help): p

Disk /dev/hda: 16 heads, 38 sectors, 683 cylinders
Units = cylinders of 608 * 512 bytes

   Device Boot  Begin   Start     End  Blocks   Id  System
/dev/hda1   *       1       1     203   61693    6  DOS 16-bit >=32M
/dev/hda2         204     204     473   82080   81  Linux/MINIX
/dev/hda3         474     474     507   10336   81  Linux/MINIX
```

As you can see, `/dev/hda2` is now a partition of size 82080 blocks (which corresponds to about 80 megabytes), and `/dev/hda3` is 10336 blocks (about 10 megs).

Finally, we use the `w` command to write the changes to disk and exit `fdisk`.

```
Command (m for help): w

#
```

Keep in mind that none of the changes you make while running `fdisk` will take effect until you give the `w` command, so you can toy with different configurations and only save them when you're done. Also, if you want to quit `fdisk` at any time without saving the changes, use the `q` command. Also remember that you shouldn't modify partitions for any operating systems other than Linux with the Linux `fdisk` program.

3.3.5 Rebooting the system

Before you create your filesystems, you need to reboot the system in order for the changes to the partition table to take effect. (Note that newer versions of the Linux `fdisk` program automatically reset the partition table when exiting; this means that you *don't* have to reboot. If `fdisk` prints a message such as "`Partition table synchronized`", then there is no need to reboot, and you can skip to the next section).

You should boot the system from the diskette or CD-ROM for your distribution of Linux, as before. (You can't boot from the hard drive, yet, because you haven't installed the Linux software).

Under Linux, you should *never* reboot the system with ctrl-alt-del (the "Vulcan Nerve Pinch") or by pressing the reset button on your system. This is because Linux, like other versions of UNIX,

buffers disk I/O in memory. The system only writes the memory buffers to disk every 30 seconds or so. If you were to reset the system before it was able to "sync" and write the information to disk, you would corrupt the data on your hard drive.

Therefore, the `shutdown` command is usually used to reboot the system. (See Section 5.3.) However, most Linux distributions do not provide the `shutdown` command until you have installed the software; the best you can do at this point is use reboot your system as you normally would (with the reset button). Just keep in mind that after you have installed the Linux software, you should never reboot in this way.

3.3.6 Making the swap space

After you have created your Linux partitions, you are ready to "format" them by creating a **filesystem** on the root partition, and preparing the swap space on the swap partition.

First, let's create the swap space. Reboot the system, as described above, and login as `root`. The command used to prepare swap space is `mkswap`, and it takes the form

`mkswap -c` ⟨*partition*⟩ ⟨*size*⟩

where ⟨*partition*⟩ is the name of the swap partition you have created, and ⟨*size*⟩ is the size of the partition, in blocks[6]. For example, if your swap partition is on `/dev/hda3` and is 10336 blocks in size, use the command

`#` *mkswap -c /dev/hda3 10336*

The `-c` option tells `mkswap` to check for bad blocks on the partition when creating the swap space.

3.3.7 Creating the filesystems

As mentioned previously, before you may use your Linux root partition to store files, you must create a **filesystem** on it. There are several types of filesystems available for Linux. The most commonly used filesystem type is the **Linux Second Extended Filesystem**, or *ext2fs*—it is used by the majority of Linux users worldwide. Because of the ext2fs's popularity, speed, and robustness, we'll only cover the creation of an ext2fs-type filesystem here[7].

To create a filesystem on the Linux root partition, use the command

`mke2fs -c` ⟨*partition*⟩ ⟨*size*⟩

where ⟨*partition*⟩ is the name of the Linux root partition, and ⟨*size*⟩ is the size of the partition in blocks. For example, to create a 82080-block filesystem on `/dev/hda2`, use the command

[6] This is the size as reported by `fdisk`, using the `p` menu option.

[7] If you're interested in using a filesystem type other than the ext2fs, see the Linux FAQ. Information on obtaining the FAQ is found in Appendix B.

```
mke2fs -c /dev/hda2 82080
```

If you're creating more than one filesystem for Linux (see Section A.3), you'll need to use the appropriate `mke2fs` command for those partitions as well.

3.4 Installing SLS

The `doinstall` command is used to install SLS. It is relatively straightforward, and prompts you for a number of options before installing.

3.4.1 The SLS bootdisk

Before you begin, make sure you have in hand an MS-DOS formatted floppy, which will be used during the installation process to create a Linux boot disk: essentially, a copy of the Linux kernel used to boot your system[8].

3.4.2 Using `doinstall`

The syntax of the `doinstall` command is

`doinstall` ⟨*partition*⟩ [⟨*partition*⟩ ⟨*directory*⟩ ...]

The first ⟨*partition*⟩ is required: it is the partition which the root filesystem is mounted on (such as `/dev/hda2`). The other arguments are used to tell `doinstall` where to mount any additional filesystems if you're using them. See Section A.3 for details on this procedure.

For example, if your Linux root filesystem is on `/dev/hda2`, use the command

```
softland:/# doinstall /dev/hda2
```

The `doinstall` command will step you through the installation process. It will ask you what media to install from (e.g., floppy, hard drive, CD-ROM, or network). Choose the appropriate option (which is probably "floppy", unless you're installing from CD-ROM or the hard drive. See Sections A.1 and A.2.)

`doinstall` will also ask you how much of the software to install, depending on which disk sets you're installing.

Tiny base system	`a` disk series only (15 megs)
Main base system	`a`, `b`, and `c` series (45 megs)
Main base system + X11	`a`, `b`, `c`, and `x` series (70 megs)
Full system	`a`, `b`, `c`, `x`, `d`, `s`, and `t` series (90 megs)

[8] In Section 5.2.2, we'll cover how to install LILO to boot your system from the hard drive. Booting Linux from floppy is no big deal; once it's finished booting control is transferred to the hard drive, and the floppy isn't accessed again.

You can selectively install packages from the disk series during the installation process—you need not install all of the software on each disk series. Just answer "`yes`" to the question "`Prompt for each package?`" when you run `doinstall`.

During the installation, if you receive numerous error messages such as "`No space left on device`", then you didn't create filesystems large enough to hold the software. You should go back and either repartition your drive to allocate more space, or being the installation again (this time not installing as much of the software). In any case, if you choose to reinstall SLS for some reason, it is important that you redo the `mke2fs` command (i.e., re-create your filesystems). This will delete any existing software on your Linux partitions before you attempt to reinstall.

If everything goes well, then congratulations! You've just installed Linux on your system. Go have a Diet Coke or something—you deserve it.

3.4.3 Rebooting the system

To boot Linux, simply boot the Linux boot disk that was created during the installation procedure (not the `a1` disk). Also, it's a good idea to keep your SLS installation disks around—if you accidentally trash your system, for example, you can use the SLS `a1` disk to recover your files (with a little luck).

3.5 Now That You Have Linux Installed

You've probably run into a few problems along the way while trying to install Linux. The most common problem is having corrupt files on your installation disks, caused by either a mistake while downloading or a mistake by the folks who put together the release you're trying to install. Hopefully, however, all has gone well and you now have a running Linux system.

The next few chapters will help you get aquainted with the Linux system.

Chapter 4

Linux Tutorial

4.1 Introduction

New users of UNIX and Linux may be a bit intimidated by the size and apparent complexity of the system before them. There are many good books on using UNIX out there, for all levels of expertise from novice to expert. However, none of these books covers, specifically, an introduction to using Linux. While 95% of using Linux is exactly like using other UNIX systems, the most straightforward way to get going on your new system is with a tutorial tailored for Linux. Herein is such a tutorial.

This chapter does not go into a large amount of detail or cover many advanced topics. Instead, it is intended to get the new Linux user running, on both feet, so that he or she may then read a more general book about UNIX and understand the basic differences between other UNIX systems and Linux.

Very little is assumed here, except perhaps some familiarity with personal computer systems, and MS-DOS. However, even if you're not an MS-DOS user, you should be able to understand everything here. At first glance, UNIX looks a lot like MS-DOS (after all, MS-DOS was modeled on the CP/M operating system, which in turn was modeled on UNIX). However, only the very superficial features of UNIX resemble MS-DOS in any way. Even if you're completely new to the PC world, this tutorial should be of help.

And, before we begin: *Don't be afraid to experiment.* The system won't bite you. You can't destroy anything by working on the system. UNIX has some amount of security built in, to prevent "normal" users (the role which you will now assume) from damaging files which are essential to the system. Even so, the absolute worst thing that can happen is that you'll delete all of your files—and you'll have to go back and re-install the system. So, at this point, you have nothing to lose.

4.2 Basic UNIX Concepts

UNIX is a multitasking, multiuser operating system. This means that there can be many people using one computer at the same time, running many different applications. (This differs from MS-

DOS, where only one person can use the system at any one time.) Under UNIX, for users to identify themselves to the system, they must **log in**, which entails two steps: Entering your **login name** (the name which the system identifies you as), and entering your **password**, which is your personal secret key to logging into your account. Because only you know your password, no one else can login to the system under your username.

On traditional UNIX systems, the system administrator will assign you a username and an initial password when you are given an account on the system. However, because you are the system administrator, you must set up your own account before you can login—see Section 4.2.1, below. For the following discussions, we'll use the imaginary username "`larry`".

In addition, each UNIX system has a **hostname** assigned to it. It is this hostname that gives your machine a name, gives it character and charm. The hostname is used to identify individual machines on a network, but even if your machine isn't networked, it should have a hostname. In Section 5.9.2 we'll cover setting your system's hostname. For our examples, below, the system's hostname is "`mousehouse`".

4.2.1 Creating an account

Before you can use the system, you must set up a user account for yourself. This is because it's usually not a good idea to use the `root` account for normal use. The `root` account should be reserved for running priveleged commands and for maintaining the system, as discussed in Section 5.1.

In order to create an account for yourself, you need to login as `root` and use the `useradd` or `adduser` command. See Section 5.4 for information on this procedure.

4.2.2 Logging in

At login time, you'll see a prompt resembling the following on your screen:

```
mousehouse login:
```

Here, enter your username, and press the Return key. Our hero, `larry`, would type the following:

```
mousehouse login:  larry
Password:
```

Now, enter your password. It won't be echoed to the screen when you login, so type carefully. If you mistype your password, you'll see the message

```
Login incorrect
```

and you'll have to try again.

Once you have correctly entered the username and password, you are officially logged into the system, and are free to roam.

4.2.3 Virtual consoles

The system's **console** is the monitor and keyboard connected directly to the system. (Because UNIX is a multiuser operating system, you may have other terminals connected to serial ports on your system, but these would not be the console.) Linux, like some other versions of UNIX, provides access to **virtual consoles** (or VC's), which allow you to have more than one login session from your console at a time.

To demonstrate this, login to your system (as demonstrated above). Now, press alt-F2. You should see the `login:` prompt again. You're looking at the second virtual console—you logged into the first. To switch back to the first VC, press alt-F1. *Voila!* You're back to your first login session.

A newly-installed Linux system probably allows you to access the first four VC's, using alt-F1 through alt-F4. However, it is possible to enable up to 12 VC's—one for each function key on your keyboard. As you can see, use of VC's can be very powerful—you can be working on several different VC's at once.

While the use of VC's is somewhat limiting (after all, you can only be looking at one VC at a time), it should give you a feel for UNIX's multiuser capabilities. While you're working on VC #1, you can switch over to VC #2 and start working on something else.

4.2.4 Shells and commands

For most of your explorations in the world of UNIX, you'll be talking to the system through the use of a **shell**. A shell is just a program which takes user input (e.g., commands which you type) and translates them into instructions. This can be compared to the `COMMAND.COM` program under MS-DOS, which does essentially the same thing. The shell is just one interface to UNIX. There are many possible interfaces—such as the X Window System, which lets you run commands by using the mouse and keyboard in conjunction.

As soon as you login, the system starts the shell, and you can type commands to it. Here's a quick example. Here, Larry logs in, and is left sitting at the shell **prompt**.

```
mousehouse login:  larry
Password:  larry's password
Welcome to Mousehouse!

/home/larry#
```

"`/home/larry#`" is the shell's prompt, indicating that it's ready to take commands. (More on what the prompt itself means later.) Let's try telling the system to do something interesting:

```
/home/larry# make love
make:  *** No way to make target 'love'.  Stop.
/home/larry#
```

Well, as it turns out `make` was the name of an actual program on the system, and the shell executed this program when given the command. (Unfortunately, the system was being unfriendly.)

This brings us to one burning question: What are commands? What happens when you type "`make love`"? The first word on the command line, "`make`", is the name of the command to be executed. Everything else on the command line is taken as arguments to this command. Examples:

```
/home/larry# cp foo bar
```

Here, the name of the command is "`cp`", and the arguments are "`foo`" and "`bar`".

When you type a command, the shell does several things. First of all, it looks at the command name, and checks to see if it is a command which is internal to the shell. (That is, a command which the shell knows how to execute itself. There are a number of these commands, and we'll go into them later.) The shell also checks to see if the command is an alias, or substitute name, for another command. If neither of these conditions apply, the shell looks for a program, on the disk, with the command's name. If it finds such a program, the shell runs it, giving the program the arguments specified on the command line.

In our example, the shell looks for the program called `make`, and runs it with the argument `love`. `Make` is a program often used to compile large programs, and it takes as arguments the name of a "target" to compile. In the case of "`make love`", we instructed `make` to compile the target `love`. Because `make` can't find a target by this name, it fails with a humorous error message, and we are returned to the shell prompt.

What happens if we type a command to a shell, and the shell can't find a program with the command name to run? Well, we can try it:

```
/home/larry# eat dirt
eat:  command not found
/home/larry#
```

Quite simply, if the shell can't find a program with the name given on the command line (here, "`eat`"), it prints an error message which should be self-explanatory. You'll often see this error message if you mistype a command (for example, if you had typed "`mkae love`" instead of "`make love`").

4.2.5 Logging out

Before we delve much further, we should tell you how to log out of the system. At the shell prompt, use the command

```
/home/larry# exit
```

to logout. There are other ways of logging out as well, but this is the most foolproof one.

4.2.6 Changing your password

You should also be aware of how to change your password. The command `passwd` will prompt you for your old password, and your new password. It will ask you to reenter the new password for validation. Be careful not to forget your password—if you do, you will have to ask the system administrator to reset it for you. (If you're the system administrator, see Section 5.4.)

4.2.7 Files and directories

Under most operating systems (UNIX included), there is the concept of a **file**, which is just a bundle of information which is given a name (called a **filename**). Examples of files would be your history term paper, an e-mail message, or an actual program which can be executed. Essentially, anything which is saved on disk is saved in an individual file.

Files are identified by their filenames. For example, the file containing your history paper might be saved with the filename `history.txt`. These names usually identify the file and its contents in some form which is meaningful to you.

With the concept of files comes the concept of directories. A **directory** is just a collection of files. It can be thought of as a "folder" which contains many different files. Directories themselves are given names, with which you can identify them. Furthermore, directories are maintained in a tree-like structure; that is, directories may contain other directories.

A file may be referred to by its **pathname**, which is made up of the filename, preceded by the name of the directory which contains the file. For example, let's say that Larry has a directory called `papers`, which contains three files: `history-final`, `english-lit`, and `masters-thesis`. (Each of these three files contains information for three of Larry's ongoing projects.) To refer to the file `english-lit`, Larry can specify the file's pathname:

```
papers/english-lit
```

As you can see, the directory and file names are separated by a single slash (/). MS-DOS users will find this convention familiar, although in the MS-DOS world, the backslash (\) is used instead.

As mentioned, directories can be nested within each other as well. For example, let's say that Larry has another directory, within `papers`, called `notes`. This directory contains the files `math-notes` and `cheat-sheet`. The pathname of the file `cheat-sheet` would be

```
papers/notes/cheat-sheet
```

Therefore, the pathname really is a "path" which you take to locate a certain file. The directory above a given subdirectory is known as the **parent directory**. Here, the directory `papers` is the parent of the `notes` directory.

4.2.8 The directory tree

Most UNIX systems have a standard layout for files, so that system resources and programs can be easily located. This layout forms a directory tree, which starts at the "/" directory, also known as "the root directory". Directly underneath / are some important subdirectories: `/bin`, `/etc`, `/dev`, and `/usr`, among others. These directories in turn contain other directories which contain system configuration files, programs, and so on.

In particular, each user has a **home directory**, which is the directory set aside for that user to store his or her files. In the examples above, all of Larry's files (such as `cheat-sheet` and `history-final`) were contained in Larry's home directory. Usually, user home directories are contained under `/home`, and are named for the user who owns that directory. Therefore, Larry's home directory is `/home/larry`.

In Figure 4.2.8 a sample directory tree is represented. It should give you some idea of how the directory tree on your system is organized.

4.2.9 The current working directory

At any given time, commands that you type to the shell are given in terms of your **current working directory**. You can think of your working directory as the directory in which you are currently "located". When you first login, your working directory is set to your home directory—`/home/larry` in our case. Whenever you reference a file, you may refer to it in relationship to your current working directory, instead of specifying the full pathname of the file.

Here's an example. Larry has the directory `papers`, and `papers` contains the file `history-final`. If Larry wants to look at this file, he can use the command

```
/home/larry# more /home/larry/papers/history-final
```

The `more` command simply displays a file, one screen at a time. However, because Larry's current working directory is `/home/larry`, he can instead refer to the file *relative* to his current location. The command would be

```
/home/larry# more papers/history-final
```

Therefore, if you begin a filename (such as `papers/final`) with a character other than "/", the system assumes that you're referring to the file in terms relative to your current working directory. This is known as a **relative pathname**.

On the other hand, if you begin a filename with a "/", the system interprets this as a full pathname—that is, a pathname including the entire path to the file, starting from the root directory, /. This is known as an **absolute pathname**.

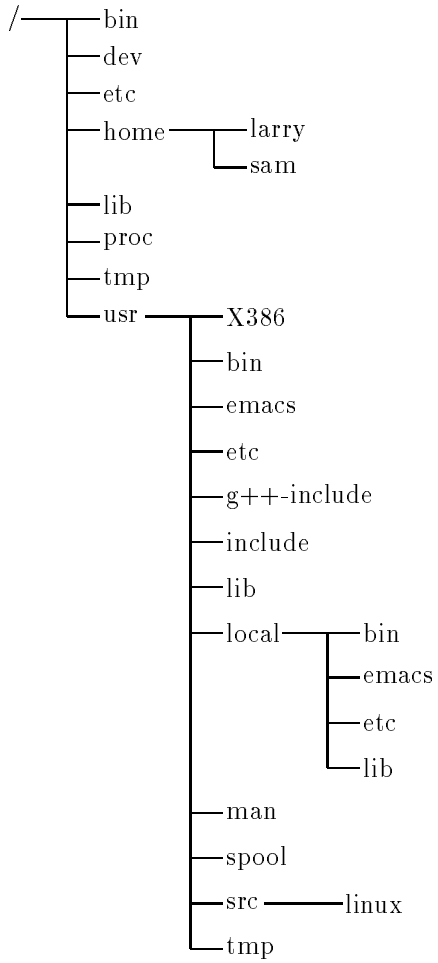

Figure 4.1: A typical (abridged) Unix directory tree.

4.2.10 Referring to home directories

Under both Tcsh and Bash on Linux, your home directory can be referred to using the tilde character ("~"). For example, the command

```
/home/larry# more ~/papers/history-final
```

is equivalent to

```
/home/larry# more /home/larry/papers/history-final
```

The "~" character is simply replaced with the name of your home directory by the shell.

In addition, you can specify other user's home directories with the tilde as well. The pathname "`~karl/letters`" translates to "`/home/karl/letters`" by the shell (if `/home/karl` is karl's home

directory). The use of the tilde is simply a shortcut; there is no directory named "~"—it's just syntactic sugar provided by the shell.

4.3 First Steps into UNIX

Before we begin, it is important to note that all file and command names on a UNIX system are case-sensitive (unlike operating systems such as MS-DOS). For example, the command `make` is very different than `Make` or `MAKE`. The same hold for file and directory names.

4.3.1 Moving around

Now that we can login, and know how to refer to files using pathnames, how can we change our current working directory, to make life easier?

The command for moving around in the directory structure is `cd`, short for "change directory". You'll notice that many often-used Unix commands are two or three letters. The usage of the `cd` command is:

`cd` ⟨*directory*⟩

where ⟨*directory*⟩ is the name of the directory which you wish to change to.

As we said, when you login, you begin in your home directory. If Larry wanted to move down into the `papers` subdirectory, he'd use the command

```
/home/larry# cd papers
/home/larry/papers#
```

As you can see, Larry's prompt changes to reflect his current working directory (so he knows where he is). Now that he's in the `papers` directory, he can look at his history final with the command

```
/home/larry/papers# more history-final
```

Now, Larry is stuck in the `papers` subdirectory. To move back up to the parent directory, use the command

```
/home/larry/papers# cd ..
/home/larry#
```

(Note the space between the "`cd`" and the "`..`".) Every directory has a subdirectory named "`..`" which refers to the parent directory. Similarly, every directory has a subdirectory named "`.`" which refers to itself. Therefore, the command

```
/home/larry/papers# cd .
```

gets us nowhere.

You can also use absolute pathnames in the `cd` command. To `cd` into Karl's home directory, we can use the command

```
/home/larry/papers# cd /home/karl
/home/karl#
```

Also, using `cd` with no argument will return you to your own home directory.

```
/home/karl# cd
/home/larry#
```

4.3.2 Looking at the contents of directories

Now that you know how to move around directories you probably think, "So what?" The basic skill of moving around directories is fairly useless, so let's introduce a new command, `ls`. `ls` prints a listing of files and directories, by default from your current directory. For example:

```
/home/larry# ls
Mail
letters
papers
/home/larry#
```

Here we can see that Larry has three entries in his current directory: `Mail`, `letters`, and `papers`. This doesn't tell us much—are these directories or files? We can use the `-F` option on the `ls` command to tell us more.

```
/home/larry# ls -F
Mail/
letters/
papers/
/home/larry#
```

From the `/` appended to each filename, we know that these three entries are in fact subdirectories.

Using `ls -F` may also append "`*`" to the end of a filename. This indicates that the file is an **executable**, or a program which can be run. If nothing is appended to the filename using `ls -F`, the file is a "plain old file", that is, it's neither a directory, or an executable.

In general, each UNIX command may take a number of options in addition to other arguments. These options usually begin with a "`-`", as demonstrated above with `ls -F`. The `-F` option tells `ls` to give more information about the type of the files involved—in this case, printing a `/` after each directory name.

If you give `ls` a directory name, it will print the contents of that directory.

```
/home/larry# ls -F papers
english-lit
history-final
masters-thesis
notes/
/home/larry#
```

Or, for a more interesting listing, let's see what's in the system's `/etc` directory.

```
/home/larry# ls /etc

Images          ftpusers        lpc             rc.new          shells
adm             getty           magic           rc0.d           startcons
bcheckrc        gettydefs       motd            rc1.d           swapoff
brc             group           mount           rc2.d           swapon
brc~            inet            mtab            rc3.d           syslog.conf
csh.cshrc       init            mtools          rc4.d           syslog.pid
csh.login       init.d          pac             rc5.d           syslogd.reload
default         initrunlvl      passwd          rmt             termcap
disktab         inittab         printcap        rpc             umount
fdprm           inittab.old     profile         rpcinfo         update
fstab           issue           psdatabase      securetty       utmp
ftpaccess       lilo            rc              services        wtmp
/home/larry#
```

(For those MS-DOS users out there, notice how the filenames can be longer than 8 characters, and can contain periods in any position. It is even possible to have more than one period in a filename.)

Let's `cd` up to the top of the directory tree, using "`cd ..`", and then down to another directory: `/usr/bin`.

```
/home/larry# cd ..
/home# cd ..
/# cd usr
/usr# cd bin
/usr/bin#
```

You can also move into directories in multiple steps, as in `cd /usr/bin`.

Try moving around various directories, using `ls` and `cd`. In some cases, you may run into a foreboding "`Permission denied`" error message. This is simply the concept of UNIX security kicking in: in order to `ls` or to `cd` into a directory, you must have permission to do so. We'll talk more about this in Section 4.9.

4.3.3 Creating new directories

It's time to learn how to create directories. This involves the use of the `mkdir` command. Try the following:

```
/home/larry# mkdir foo
/home/larry# ls -F
Mail/
foo/
letters/
papers/
/home/larry# cd foo
/home/larry/foo# ls
/home/larry/foo#
```

Congrats! You've just made a new directory and moved into it. Since there aren't any files in this new directory, let's learn how to copy files from one place to another.

4.3.4 Copying files

Copying files is done with the command `cp`:

```
/home/larry/foo# cp /etc/termcap .
/home/larry/foo# cp /etc/shells .
/home/larry/foo# ls –F
shells     termcap
/home/larry/foo# cp shells bells
/home/larry/foo# ls –F
bells     shells     termcap
/home/larry/foo#
```

The `cp` command copies the files listed on the command line to the file or directory given as the last argument. Notice how we use the directory "." to refer to the current directory.

4.3.5 Moving files

A new command named `mv` moves files, instead of copying them. The syntax is very straightforward.

```
/home/larry/foo# mv termcap sells
/home/larry/foo# ls -F
bells     sells     shells
/home/larry/foo#
```

Notice how `termcap` no longer exists, but in its place is the file `sells`. This can be used to rename files, as we have just done, but also to move a file to a completely new directory.

◇ **Note:** `mv` and `cp` will overwrite the destination file (if it already exists) without asking you. Be careful when you move a file into another directory: there may already be a file with the same name in that directory, which you'll overwrite!

4.3.6 Deleting files and directories

You now have an ugly rhyme developing with the use of the `ls` command. To delete a file, use the `rm` command. ("`rm`" stands for "remove").

```
/home/larry/foo# rm bells sells
/home/larry/foo# ls -F
shells
/home/larry/foo#
```

We're left with nothing but shells, but we won't complain. Note that `rm` by default won't prompt you before deleting a file—so be careful.

A related command to `rm` is `rmdir`. This command deletes a directory, but only if the directory is empty. If the directory contains any files or subdirectories, `rmdir` will complain.

4.3.7 Looking at files

The commands `more` and `cat` are used for viewing the contents of files. `more` displays a file, one screenful at a time, while `cat` displays the whole file at once.

To look at the file `shells`, we can use the command

```
/home/larry/foo# more shells
```

In case you're interested what `shells` contains, it's a list of valid shell programs on your system. On most systems, this includes `/bin/sh`, `/bin/bash`, and `/bin/csh`. We'll talk about these different types of shells later.

While using `more`, press Space to display the next page of text, and b to display the previous page. There are other commands available in `more` as well, these are just the basics. Pressing q will quit `more`.

Quit `more` and try `cat /etc/termcap`. The text will probably fly by much too quickly for you to read it. The name "`cat`" actually stands for "concatenate", which is the real use of the program. The `cat` command can be used to concatenate the contents of several files and save the result to another file. This will be discussed later.

4.3.8 Getting online help

Almost every UNIX system, Linux included, provides a facility known as "manual pages", or "man pages" for short. These man pages contain online documentation for all of the various system commands, resources, configuration files, and so on.

The command used to access man pages is `man`. For example, if you're interested in finding out about the other options of the `ls` command, you can type

```
/home/larry# man ls
```

and the man page for `ls` will be displayed.

Unfortunately, most of the man pages out there are written for those who already have some idea of what the command or resource does. For this reason, man pages usually only contain the hardcore technical details of the command, without a lot of tutorial. However, man pages can be an invaluable resource for jogging your memory if you forget the syntax of a command. Man pages will also tell you a lot about the commands which we won't tell you in this book.

I suggest that you try `man` for the commands we've already gone over, and whenever I introduce a new command. You'll notice some of these commands won't have man pages. This could be for several reasons. For one, the man pages haven't been written yet (the Linux Documentation Project is responsible for man pages under Linux as well. We are gradually accumulating most of the man pages available for the system). Secondly, the the command might be an internal shell command, or an alias (as discussed in Section 4.2.4), in which case it would not have a man page of its own. One example is `cd`, which is a shell internal command. The shell actually processes the `cd`—there is no separate program which contains this command.

4.4 Summary of Basic Commands

This section introduces some of the most useful basic commands on a UNIX system, including those covered in the last section.

Note that options usually begin with a "`-`", and in most cases multiple one-letter options may be combined using a single "`-`". For example, instead of using the command `ls -l -F`, it is adequate to use `ls -lF`.

Instead of listing all of the options available for each of these commands, we'll only talk about those which are useful or important at this time. In fact, most of these commands have a large number of options (most of which you'll never use). You can use `man` to see the manual pages for each command, which list all of the available options.

Also note that many of these commands take a list of files or directories as arguments, denoted by "⟨*file1*⟩ ...⟨*fileN*⟩". For example, the `cp` command takes as arguments a list of files to copy, followed by the destination file or directory. When copying more than one file, the destination must be a directory.

`cd` Change the current working directory.
Syntax: `cd` ⟨*directory*⟩
⟨*directory*⟩ is the directory to change to. ("`.`" refers to the current directory, "`..`" the parent directory.)
Example: `cd ../foo` sets the current directory to `../foo`.

`ls` Displays information about the named files and directories.
Syntax: `ls` ⟨*file1*⟩ ⟨*file2*⟩ ...⟨*fileN*⟩
Where ⟨*file1*⟩ through ⟨*fileN*⟩ are the filenames or directories to list. Options: There are more options than you want to think about. The most commonly used are `-F` (used to display some information about the type of the file), and `-l` (gives a "long" listing including file size, owner, permissions, and so on. This will be covered in detail later.)
Example: `ls -lF /home/larry` will display the contents of the directory `/home/larry`.

`cp` Copies file(s) to another file or directory.
Syntax: `cp` ⟨*file1*⟩ ⟨*file2*⟩ ...⟨*fileN*⟩ ⟨*destination*⟩
Where ⟨*file1*⟩ through ⟨*fileN*⟩ are the files to copy, and ⟨*destination*⟩ is the destination file or directory.
Example: `cp ../frog joe` copies the file `../frog` to the file or directory `joe`.

`mv` Moves file(s) to another file or directory. This command does the equivalent of a copy followed by the deletion of the original. This can be used to rename files, as in the MS-DOS command `RENAME`.
Syntax: `mv` ⟨*file1*⟩ ⟨*file2*⟩ ...⟨*fileN*⟩ ⟨*destination*⟩
Where ⟨*file1*⟩ through ⟨*fileN*⟩ are the files to move, and ⟨*destination*⟩ is the destination file or directory.
Example: `mv ../frog joe` moves the file `../frog` to the file or directory `joe`.

`rm` Deletes files. Note that when files are deleted under UNIX, they are unrecoverable (unlike MS-DOS, where you can usually "undelete" the file).
Syntax: `rm` ⟨*file1*⟩ ⟨*file2*⟩ ...⟨*fileN*⟩
Where ⟨*file1*⟩ through ⟨*fileN*⟩ are the filenames to delete.
Options: `-i` will prompt for confirmation before deleting the file.
Example: `rm -i /home/larry/joe /home/larry/frog` deletes the files `joe` and `frog` in `/home/larry`.

`mkdir` Creates new directories.
Syntax: `mkdir` ⟨*dir1*⟩ ⟨*dir2*⟩ ...⟨*dirN*⟩
Where ⟨*file1*⟩ through ⟨*fileN*⟩ are the directories to create.
Example: `mkdir /home/larry/test` creates the directory `test` under `/home/larry`.

`rmdir` This command deletes empty directories. When using `rmdir`, your current working directory must not be within the directory to be deleted.

Syntax: `rmdir` ⟨*dir1*⟩ ⟨*dir2*⟩ ...⟨*dirN*⟩
Where ⟨*file1*⟩ through ⟨*fileN*⟩ are the directories to delete.
Example: `rmdir /home/larry/papers` deletes the directory `/home/larry/papers`, if it is empty.

`man` Displays the manual page for the given command or resource (that is, any system utility which isn't a command, such as a library function.) Syntax: `man` ⟨*command*⟩
Where ⟨*command*⟩ is the name of the command or resource to get help on.
Example: `man ls` gives help on the `ls` command.

`more` Displays the contents of the named files, one screenful at a time.
Syntax: `more` ⟨*file1*⟩ ⟨*file2*⟩ ...⟨*fileN*⟩
Where ⟨*file1*⟩ through ⟨*fileN*⟩ are the files to display.
Example: `more papers/history-final` displays the file `papers/history-final`.

`cat` Officially used to concatenate files, `cat` is also used to display the entire contents of a file at once.
Syntax: `cat` ⟨*file1*⟩ ⟨*file2*⟩ ...⟨*fileN*⟩
Where ⟨*file1*⟩ through ⟨*fileN*⟩ are the files to display.
Example: `cat letters/from-mdw` displays the file `letters/from-mdw`.

`echo` Simply echoes the given arguments.
Syntax: `echo` ⟨*arg1*⟩ ⟨*arg2*⟩ ...⟨*argN*⟩
Where ⟨*arg1*⟩ through ⟨*argN*⟩ are the arguments to echo.
Example: `echo "Hello world"` displays the string "`Hello world`".

`grep` Display all of the lines in the named file(s) matching the given pattern.
Syntax: `grep` ⟨*pattern*⟩ ⟨*file1*⟩ ⟨*file2*⟩ ...⟨*fileN*⟩
Where ⟨*pattern*⟩ is a regular expression pattern, and ⟨*file1*⟩ through ⟨*fileN*⟩ are the files to search.
Example: `grep loomer /etc/hosts` will display all lines in the file `/etc/hosts` which contain the pattern "`loomer`".

4.5 Exploring the File System

The **file system** is the collection of files and the hierarchy of directories on your system. I promised before to escort you around the filesystem and the time has come.

You have the skills and the knowledge to make sense out of what I'm saying, and you have a roadmap. (Refer to Figure 4.2.8 on page 35).

First, change to the root directory (`cd /`), and do an `ls -F`. You'll probably see these directories[1]: `bin`, `dev`, `etc`, `home`, `install`, `lib`, `mnt`, `proc`, `root`, `tmp`, `user`, `usr`, and `var`.

Let's take a look at each of these directories.

[1] You may see others, and you might not see all of them. Don't worry. Every release of Linux differs in some respects.

`bin`

`/bin` is short for "binaries", or executables. This is where many essential system programs reside. Use the command "`ls -F /bin`" to list the files here. If you look down the list you may see a few commands that you recognize, such as `cp`, `ls`, and `mv`. These are the actual programs for these commands. When you use the `cp` command, you're running the program `/bin/cp`.

Using `ls -F`, you'll see that most (if not all) of the files in `/bin` have an asterisk ("`*`") appended to their filenames. This indicates that the files are executables, as described in Section 4.3.2.

`/dev`

Next on our stop is `/dev`. Take a look, again with `ls -F`.

The "files" in `/dev` are known as **device drivers**—they are used to access system devices and resources, such as disk drives, modems, memory, and so on. For example, just as you can read data from a file, you can read input from the mouse by accessing `/dev/mouse`.

The filenames beginning with `fd` are floppy disk devices. `fd0` is the first floppy disk drive, `fd1` the second. Now, the astute among you will notice that there are more floppy disk devices then just the two I've listed above: they represent specific types of floppy disks. For example, `fd1H1440` will access high-density, 3.5" diskettes in drive 1.

Here is a list of some of the most commonly used device files. Note that even though you may not have some of the devices listed below, the chances are that you'll have entries in `/dev` for them anyway.

- `/dev/console` refers to the system's console—that is, the monitor connected directly to your system.
- The various `/dev/ttyS` and `/dev/cua` devices are used for accessing serial ports. For example, `/dev/ttyS0` refers to "`COM1`" under MS-DOS. The `/dev/cua` devices are "callout" devices, which are used in conjunction with a modem.
- The device names beginning with `hd` access hard drives. `/dev/hda` refers to the *whole* first hard disk, while `hda1` refers to the first *partition* on `/dev/hda`.
- The device names beginning with `sd` are SCSI drives. If you have a SCSI hard drive, instead of accessing it through `/dev/hda`, you would access `/dev/sda`. SCSI tapes are accessed via `st` devices, and SCSI CD-ROM via `sr` devices.
- The device names beginning with `lp` access parallel ports. `/dev/lp0` refers to "`LPT1`" in the MS-DOS world.
- `/dev/null` is used as a "black hole"—any data sent to this device is gone forever. Why is this useful? Well, if you wanted to suppress the output of a command appearing on your screen, you could send that output to `/dev/null`. We'll talk more about this later.
- The device names beginning with `/dev/tty` refer to the "virtual consoles" on your system (accessed via by pressing alt-F1, alt-F2, and so on). `/dev/tty1` refers to the first VC, `/dev/tty2` refers to the second, and so on.

- The device names beginning with `/dev/pty` are "pseudo-terminals". They are used to provide a "terminal" to remote login sessions. For example, if your machine is on a network, incoming `telnet` logins would use one of the `/dev/pty` devices.

`/etc` `/etc` contains just about everything which can be labeled "et cetera"—many system configuration files, programs, and utilities. Most of the programs found in `/etc` are for use by the system administrator only. We'll cover these in depth in Chapter 5.

`/home` `/home` contains user's home directories. For example, `/home/larry` is the home directory for the user "`larry`". On a newly-installed system, there may not be any users in this directory.

`/lib` `/lib` contains **shared library images**. These files contain code which many programs share in common. Instead of each program containing its own copy of these shared routines, they are all stored in one common place, in `/lib`. This makes executable files smaller, and saves space on your system.

`/proc` `/proc` is a "virtual filesystem", the files in which are stored in memory, not on the drive. They refer to the various **processes** running on the system, and allow you to get information about what programs and processes are running at any given time. We'll go into more detail in Section 4.10.1.

`/tmp` Many programs have a need to generate some information and store it in a temporary file. The canonical location for these files is in `/tmp`.

`/usr` `/usr` is a very important directory. It contains a number of subdirectories which in turn contain some of the most important and useful programs and configuration files used on the system.

The various directories described above are essential for the system to operate, but most of the things found in `/usr` are optional for the system. However, it is those optional things which make the system useful and interesting. Without `/usr`, you'd more or less have a boring system, only with programs like `cp` and `ls`. `/usr` contains most of the larger software packages and the configuration files which accompany them.

`/usr/X386` `/usr/X386` contains The X Window System, if you installed it. The X Window System is a large, powerful graphical environment which provides a large number of graphical utilities and programs, displayed in "windows" on your screen. If you're at all familiar with the Microsoft Windows or Macintosh environments, X Windows will look very familiar. The `/usr/X386` directory contains all of the X Windows executables, configuration files, and support files. This will be covered in more detail in Section 6.1.

`/usr/adm` `/usr/adm` contains various files of interest to the system administrator, specifically system logs, which record any errors or problems with the system. Other files record logins to the system, as well as failed login attempts. This will be covered

in Chapter 5.

`/usr/bin` `/usr/bin` is the real warehouse for software on any UNIX system. It contains most of the executables for programs not found in other places, such as `/bin`.

`/usr/etc` Just as `/etc` contained miscellaneous system programs and configuration files, `/usr/etc` contains even more of these utilities and files. In general, the files found in `/usr/etc` are not essential to the system, unlike those found in `/etc`, which are.

`/usr/include` `/usr/include` contains **include files** for the C compiler. These files (most of which end in `.h`, for "header") declare data structure names, subroutines, and constants used when writing programs in C. Those files found in `/usr/include/sys` are generally used when programming on the UNIX system level. If you are familiar with the C programming language, here you'll find header files such as `stdio.h`, which declares functions such as `printf()`.

`/usr/g++-include`
`/usr/g++-include` contains include files for the C++ compiler (much like `/usr/include`).

`/usr/lib` `/usr/lib` contains the "stub" and "static" library equivalents to the files found in `/lib`. When compiling a program, the program is "linked" with the libraries found in `/usr/lib`, which then directs the program to look in `/lib` when it needs the actual code in the library. In addition, various other programs store configuration files in `/usr/lib`.

`/usr/local` `/usr/local` is a lot like `/usr`—it contains various programs and files not essential to the system, but which make the system fun and exciting. In general, those programs found in `/usr/local` are specialized for your system specifically—that is, `/usr/local` differs greatly between UNIX systems.

Here, you'll find large software packages such as TEX (a document formatting system) and Emacs (a large and powerful editor), if you installed them.

`/usr/man` This directory contains the actual man pages. There are two subdirectories for every man page "section" (use the command `man man` for details). For example, `/usr/man/man1` contains the source (that is, the unformatted original) for man pages in section 1, and `/usr/man/cat1` contains the formatted man pages for section 1.

`/usr/spool` `/usr/spool` contains files which are to be "spooled" to another program. For example, if your machine is connected to a network, incoming mail will be stored in `/usr/spool/mail`, until you read it or delete it. Outgoing or incoming news articles may be found in `/usr/spool/news`, and so on.

`/usr/src` `/usr/src` contains the source code (the uncompiled program) for various programs on your system. The most important thing here is `/usr/src/linux`, which contains the source code for the Linux kernel.

4.6 Types of shells

As I have mentioned too many times before, UNIX is a multitasking, multiuser operating system. Multitasking is *very* useful, and once you get used to it, you'll use it all of the time. Before long, you'll be able to run programs in the "background", switch between multiple tasks, and "pipeline" programs together to achieve complicated results with a single command.

Many of the features we'll be covering in this section are features provided by the shell itself. Be careful not to confuse UNIX (the actual operating system) with the shell—the shell is just an interface to the underlying system. The shell provides a great deal of functionality on top of UNIX itself.

The shell is not only an interpreter for your interactive commands, which you type at the prompt. It is also a powerful programming language, which allows you to write **shell scripts**, to "batch" several shell commands together in a file. MS-DOS users will recognize the similarity to "batch files". Use of shell scripts is a very powerful tool, which will allow you to automate and expand you usage of UNIX. See Section 4.12.1 for more information.

There are several types of shells in the UNIX world. The two major types are the "Bourne shell" and the "C shell". The Bourne shell uses a command syntax like the original shell on early UNIX systems, such as System III. The name of the Bourne shell on most UNIX systems is `/bin/sh` (where `sh` stands for "shell"). The C shell (not to be confused with sea shell) uses a different syntax, somewhat like the programming language C, and on most UNIX systems is named `/bin/csh`.

Under Linux, there are several variations of these shells available. The two most commonly used are the Bourne Again Shell, or "Bash" (`/bin/bash`), and Tcsh (`/bin/tcsh`). Bash is a form of the Bourne shell with many of the advanced features found in the C shell. Because Bash supports a superset of the Bourne shell syntax, any shell scripts written in the standard Bourne shell should work with Bash. For those who prefer to use the C shell syntax, Linux supports Tcsh, which is an expanded version of the original C shell.

The type of shell that you decide to use is mostly a religious issue. Some folks prefer the Bourne shell syntax with the advanced features of Bash, and some prefer the more structured C shell syntax. As far as normal commands, such as `cp` and `ls`, are concerned, the type of shell you're using doesn't matter—the syntax is the same. Only when you start to write shell scripts or use some of the advanced features of the shell do the differences between shell types begin to matter.

As we're discussing some of the features of the shell, below, we'll note those differences between Bourne and C shells. However, for the purposes of this manual, most of those differences are minimal. (If you're really curious at this point, read the man pages for `bash` and `tcsh`).

4.7 Wildcards

A key feature of most Unix shells is the ability to reference more than one filename using special characters. These so-called **wildcards** allow you to refer to, say, all filenames which contain the character "`n`".

The wildcard "*" refers to any character or string of characters in a filename. For example, when you use the character "*" in a filename, the shell replaces it with all possible substitutions from filenames in the directory which you're referencing.

Here's a quick example. Let's suppose that Larry has the files `frog`, `joe`, and `stuff` in his current directory.

```
/home/larry# ls
frog     joe     stuff
/home/larry#
```

To access all files with the letter "o" in the filename, we can use the command

```
/home/larry# ls *o*
frog     joe
/home/larry#
```

As you can see, the use of the "*" wildcard was replaced with all substitutions which matched the wildcard from filenames in the current directory.

The use of "*" by itself simply matches all filenames, because all characters match the wildcard.

```
/home/larry# ls *
frog     joe     stuff
/home/larry#
```

Here are a few more examples.

```
/home/larry# ls f*
frog
/home/larry# ls *ff
stuff
/home/larry# ls *f*
frog     stuff
/home/larry# ls s*f
stuff
/home/larry#
```

The process of changing a "*" into filenames is called **wildcard expansion** and is done by the shell. This is important: the individual commands, such as `ls`, *never* see the "*" in their list of parameters. The shell expands the wildcard to include all of the filenames which match. So, the command

```
/home/larry# ls *o*
```

is expanded by the shell to actually be

```
/home/larry# ls frog joe
```

One important note about the "*" wildcard. Using this wildcard will *not* match filenames which begin with a single period ("."). These files are treated as "hidden" files—while they are not really hidden, they don't show up on normal `ls` listings, and aren't touched by the use of the "*" wildcard.

Here's an example. We already mentioned that each directory has two special entries in it: "." refers to the current directory, and ".." refers to the parent directory. However, when you use `ls`, these two entries don't show up.

```
/home/larry# ls
frog     joe     stuff
/home/larry#
```

If you use the `-a` switch with `ls`, however, you can display filenames which begin with ".". Observe:

```
/home/larry# ls -a
.     ..     .bash_profile     .bashrc     frog      joe     stuff
/home/larry#
```

Now we can see the two special entries, "." and "..", as well as two other "hidden" files—`.bash_profile` and `.bashrc`. These two files are startup files used by `bash` when larry logs in. More on them in Section 4.12.3.

Note that when we use the "*" wildcard, none of the filenames beginning with "." are displayed.

```
/home/larry# ls *
frog     joe     stuff
/home/larry#
```

This is a safety feature: if the "*" wildcard matched filenames beginning with ".", it would also match the directory names "." and "..". This can be dangerous when using certain commands.

Another wildcard is "?". The "?" wildcard will only expand a single character. Thus, "`ls ?`" will display all one character filenames, and "`ls termca?`" would display "`termcap`" but *not* "`termcap.backup`". Here's another example:

```
/home/larry# ls j?e
joe
/home/larry# ls f??g
frog
/home/larry# ls ????f
stuff
/home/larry#
```

As you can see, wildcards allow you to specify many files at one time. In the simple command summary, in Section 4.4, we said that the `cp` and `mv` commands actually can copy or move multiple files at one time. For example,

```
/home/larry# cp /etc/s* /home/larry
```

will copy all filenames in `/etc` beginning with "`s`" to the directory `/home/larry`. Therefore, the format of the `cp` command is really

`cp` ⟨*file1*⟩ ⟨*file2*⟩ ⟨*file3*⟩ ... ⟨*fileN*⟩ ⟨*destination*⟩

where ⟨*file1*⟩ through ⟨*fileN*⟩ is a list of filenames to copy, and ⟨*destination*⟩ is the destination file or directory to copy them to. `mv` has an identical syntax.

Note that if you are copying or moving more than one file, the ⟨*destination*⟩ must be a directory. You can only copy or move a *single* file to another file.

4.8 UNIX Plumbing

4.8.1 Standard input and output

Many UNIX commands get input from what is known as **standard input** and send their output to **standard output** (often abbreviated as "stdin" and "stdout"). Your shell sets things up so that standard input is your keyboard, and standard output is the screen.

Here's an example using the command `cat`. Normally, `cat` reads data from all of the filenames given on the command line and sends this data directly to stdout. Therefore, using the command

```
/home/larry/papers# cat history-final masters-thesis
```

will display the contents of the file `history-final` followed by `masters-thesis`.

However, if no filenames are given to `cat` as parameters, it instead reads data from stdin, and sends it back to stdout. Here's an example.

```
/home/larry/papers# cat
Hello there.
Hello there.
Bye.
Bye.
ctrl-D
/home/larry/papers#
```

As you can see, each line that the user types (displayed in italics) is immediately echoed back by the `cat` command. When reading from standard input, commands know that the input is "finished" when they receive an EOT (end-of-text) signal. In general, this is generated by pressing ctrl-D.

Here's another example. The command `sort` reads in lines of text (again, from stdin, unless files are given on the command line), and sends the sorted output to stdout. Try the following.

```
/home/larry/papers# sort
bananas
carrots
apples
ctrl-D
apples
bananas
carrots
/home/larry/papers#
```

Now we can alphabetize our shopping list... isn't UNIX useful?

4.8.2 Redirecting input and output

Now, let's say that we wanted to send the output of `sort` to a file, to save our shopping list elsewhere. The shell allows us to **redirect** standard output to a filename, using the "`>`" symbol. Here's how it works.

```
/home/larry/papers# sort > shopping-list
bananas
carrots
apples
ctrl-D
/home/larry/papers#
```

As you can see, the result of the `sort` command isn't displayed, instead it's saved to the file `shopping-list`. Let's look at this file.

```
/home/larry/papers# cat shopping-list
apples
bananas
carrots
/home/larry/papers#
```

Now we can sort our shopping list, and save it, too! But let's suppose that we were storing our unsorted, original shopping list in the file `items`. One way of sorting the information and saving it to a file would be to give `sort` the name of the file to read, in lieu of standard input, and redirect standard output as we did above. As so:

```
/home/larry/papers# sort items > shopping-list
/home/larry/papers# cat shopping-list
apples
bananas
carrots
/home/larry/papers#
```

However, there's another way of doing this. Not only can we redirect standard output, but we can redirect standard *input* as well, using the "<" symbol.

```
/home/larry/papers# sort < items
apples
bananas
carrots
/home/larry/papers#
```

Technically, `sort < items` is equivalent to `sort items`, but the former allows us to demonstrate the point: `sort < items` behaves as if the data in the file `items` was typed to standard input. The shell handles the redirection. `sort` wasn't given the name of the file (`items`) to read; as far as `sort` is concerned, it was still reading from standard input as if you had typed the data from your keyboard.

This introduces the concept of a **filter**. A filter is a program which reads data from standard input, processes it in some way, and sends the processed data to standard output. Using redirection, standard input and/or standard output can be referenced from files. `sort` is a simple filter: it sorts the incoming data and sends the result to standard output. `cat` is even simpler: it doesn't do anything with the incoming data, it simply outputs whatever was given to it.

4.8.3 Using pipes

We've already demonstrated how to use `sort` as a filter. However, these examples assumed that you had data in a file somewhere, or were willing to type the data to standard input yourself. What if the data you wanted to sort came from the output of another command, such as `ls`? For example, using the `-r` option with `sort` sorts the data in reverse-alphabetical order. If you wanted to list the files in your current directory in reverse order, one way to do it would be:

```
/home/larry/papers# ls
english-list
history-final
masters-thesis
notes
/home/larry/papers# ls > file-list
/home/larry/papers# sort -r file-list
notes
masters-thesis
history-final
english-list
/home/larry/papers#
```

Here, we saved the output of `ls` in a file, and then ran `sort -r` on that file. But this is unwieldy and causes us to use a temporary file to save the data from `ls`.

The solution is to use **pipelining**. Pipelining is another feature of the shell which allows you to connect a string of commands in a "pipe", where the stdout of the first command is sent directly to

the stdin of the second command, and so on. Here, we wish to send the stdout of `ls` to the stdin of `sort`. The "|" symbol is used to create a pipe:

```
/home/larry/papers# ls | sort -r
notes
masters-thesis
history-final
english-list
/home/larry/papers#
```

This command is much shorter, and obviously easier to type.

Another useful example—using the command

```
/home/larry/papers# ls /usr/bin
```

is going to display a long list a files, most of which will fly past the screen too quickly for you to read them. Instead, let's use `more` to display the list of files in `/usr/bin`.

```
/home/larry/papers# ls /usr/bin | more
```

Now you can page down the list of files at your own leisure.

But the fun doesn't stop here! We can pipe more than two commands together. The command `head` is a filter which displays the first lines from an input stream (here, input from a pipe). If we wanted to display the last filename in alphabetical order in the current directory, we can use:

```
/home/larry/papers# ls | sort -r | head -1
notes
/home/larry/papers#
```

where `head -1` simply displays the first line of input that it receives (in this case, the stream of reverse-sorted data from `ls`).

4.8.4 Non-destructive redirection

Using ">" to redirect output to a file is destructive: in other words, the command

```
/home/larry/papers# ls > file-list
```

overwrites the contents of the file `file-list`. If, instead, you redirect with the symbol ">>", the output will be appended to the named file, instead of overwriting it.

```
/home/larry/papers# ls >> file-list
```

will append the output of the `ls` command to `file-list`.

Just keep in mind that redirection and using pipes are features provided by the shell—the shell provides this handy syntax using ">" and ">>" and "|". It has nothing to do with the commands used themselves, but the shell.

4.9 File Permissions

4.9.1 Concepts of file permissions

Because there are multiple users on a UNIX system, in order to protect individual user's files from tampering by other users, UNIX provides a mechanism known as **file permissions**. This mechanism allows files and directories to be "owned" by a particular user. As an example, because Larry created the files in his home directory, Larry owns those files, and has access to them.

UNIX also allows files to be shared between users and groups of users. If Larry so desired, he could cut off access to his files, such that no other user could access them. However, on most systems the default is to allow other users to read your files, but not modify or delete them in any way.

As explained above, every file is owned by a particular user. However, files are also owned by a particular **group**, which is a system-defined group of users. Every user is placed into at least one group when that user is created. However, the system administrator may also grant the user access to more than one group.

Groups are usually defined by the type of users which access the machine. For example, on a university UNIX system, users may be placed into the groups `student`, `staff`, `faculty` or `guest`. There are also a few system-defined groups (such as `bin` and `admin`) which are used by the system itself to control access to resources—very rarely do actual users belong to these system groups.

Permissions fall into three main divisions: read, write, and execute. These permissions may be granted to three classes of users: the owner of the file, the group to which the file belongs, and to all users, regardless of group.

Read permission allows a user to read the contents of the file, or in the case of directories, to list the contents of the directory (using `ls`). Write permission allows the user to write to and modify the file. For directories, write permission allows the user to create new files or delete files within that directory. Finally, execute permission allows the user to run the file as a program or shell script (if the file happens to be a program or shell script, that is). For directories, having execute permission allows the user to `cd` into the directory in question.

4.9.2 Interpreting file permissions

Let's look at an example to demonstrate file permissions. Using the `ls` command with the `-l` option will display a "long" listing of the file, including file permissions.

```
/home/larry/foo# /home/larry/foo# ls -l stuff

-rw-r--r--   1 larry    users         505 Mar 13 19:05 stuff

/home/larry/foo#
```

The first field printed in the listing represents the file permissions. The third field is the owner of the file (`larry`), and the fourth field is the group to which the file belongs (`users`). Obviously, the last field is the name of the file (`stuff`), and we'll cover the other fields later.

This file is owned by `larry`, and belongs to the group `users`. Let's look at the file permissions. The string `-rw-r--r--` lists, in order, the permissions granted to the file's owner, the file's group, and everybody else.

The first character of the permissions string ("`-`") represents the type of file. A "`-`" just means that this is a regular file (as opposed to a directory or device driver). The next three letters ("`rw-`") represent the permissions granted to the file's owner, `larry`. The "`r`" stands for "read" and the "`w`" stands for "write". Thus, `larry` has read and write permission to the file `stuff`.

As we mentioned, besides read and write permission, there is also "execute" permission—represented by an "`x`". However, there is a "`-`" here in place of the "`x`", so Larry doesn't have execute permission on this file. This is fine, the file `stuff` isn't a program of any kind. Of course, because Larry owns the file, he may grant himself execute permission for the file if he so desires. This will be covered shortly.

The next three characters, `r--`, represent the group's permissions on the file. The group which owns this file is `users`. Because only an "`r`" appears here, any user which belongs to the group `users` may read this file.

The last three characters, also `r--`, represent the permissions granted to every other user on the system (other than the owner of the file and those in the group `users`). Again, because only an "`r`" is present, other users may read the file, but not write to it or execute it.

Here are some other examples of group permissions.

`-rwxr-xr-x` The owner of the file may read, write, and execute the file. Users in the file's group, and all other users, may read and execute the file.

`-rw-------` The owner of the file may read and write the file. No other user can access the file.

`-rwxrwxrwx` All users may read, write, and execute the file.

4.9.3 Dependencies

It is important to note that the permissions granted to a file also depend on the permissions of the directory in which the file is located. For example, even if a file is set to `-rwxrwxrwx`, other users cannot access the file unless they have read and execute access to the directory in which the file is located. For example, if Larry wanted to restrict access to all of his files, he could simply set the permissions on his home directory `/home/larry` to `-rwx------`. In this way, no other user has access to his directory, and all files and directories within it. Larry doesn't need to worry about the individual permissions on each of his files.

In other words, to access a file at all, you must have execute access to all directories along the file's pathname, and read access to the file itself.

Usually, users on a UNIX system are very open with their files. The usual set of permissions given to files is `-rw-r--r--`, which will allow other users to read the file, but not change it in any way. The usual set of permissions given to directories is `-rwxr-xr-x`, which will allow other users to look through your directories, but not create or delete files within them.

However, many users wish to keep other users out of their files. Setting the permissions of a file to `-rw-------` will not allow any other user to access the file. Likewise, setting the permissions of a directory to `-rwx------` will keep other users out of the directory in question.

4.9.4 Changing permissions

The command `chmod` is used to set the permissions on a file. Only the owner of a file may change the permissions on that file. The syntax of `chmod` is:

`chmod {a,u,g,o}{+,-}{r,w,x}` ⟨*filenames*⟩

Briefly, you supply one or more of **a**ll, **u**ser, **g**roup, or **o**ther. Then you specify whether you are adding rights (`+`) or taking them away (`-`). Finally, you specify one or more of **r**ead, **w**rite, and e**x**ecute. Some examples of legal commands are:

`chmod a+r stuff`
: Gives all users read access to the file.

`chmod +r stuff`
: Same as above—if none of `a`, `u`, `g`, or `o` is specified, `a` is assumed.

`chmod og-x stuff`
: Remove execute permission from users other than the owner.

`chmod u+rwx stuff`
: Allow the owner of the file to read, write, and execute the file.

`chmod o-rwx stuff`
: Remove read, write, and execute permission from users other than the owner and users in the file's group.

4.10 Job Control

4.10.1 Jobs and processes

Job control is a feature provided by many shells (Bash and Tcsh included) which allows you to control multiple running commands, or **jobs**, at once. Before we can delve much further, we need to talk about **processes**.

Every time you run a program, you start what is known as a *process*—which is just a fancy name for a running program. The command `ps` displays a list of currently running processes. Here's an example:

`/home/larry#` *ps*

```
  PID TT STAT  TIME COMMAND
   24  3 S     0:03 (bash)
  161  3 R     0:00 ps

/home/larry#
```

The `PID` listed in the first column is the **process ID**, a unique number given to every running process. The last column, `COMMAND`, is the name of the running command. Here, we're only looking at the processes which Larry is currently running[2]. These are `bash` (Larry's shell), and the `ps` command itself. As you can see, `bash` is running concurrently with the `ps` command. `bash` executed `ps` when Larry typed the command. After `ps` is finished running (after the table of processes is displayed), control is returned to the `bash` process, which displays the prompt, ready for another command.

A running process is known as a *job* to the shell. The terms *process* and *job* are interchangeable. However, a process is usually referred to as a "job" when used in conjunction with **job control**—a feature of the shell which allows you to switch between several independent jobs.

In most cases users are only running a single job at a time—that being whatever command they last typed to the shell. However, using job control, you can run several jobs at once, switching between them as needed. How might this be useful? Let's say that you're editing a text file and need to suddenly interrupt your editing and do something else. With job control, you can temporarily suspend the editor, and back at the shell prompt start to work on something else. When you're done, you can start the editor back up, and be back where you started, as if you never left the editor. This is just one example. There are many practical uses for job control.

4.10.2 Foreground and background

Jobs can either be in the **foreground** or in the **background**. There can only be one job in the foreground at any one time. The foreground job is the job which you interact with—it receives input from the keyboard and sends output to your screen. (Unless, of course, you have redirected input or output, as described in Section 4.8). On the other hand, jobs in the background do not receive input from the terminal—in general, they run along quietly without need for interaction.

Some jobs take a long time to finish, and don't do anything interesting while they are running. Compiling programs is one such job, as is compressing a large file. There's no reason why you should sit around being bored while these jobs complete their tasks; you can just run them in the background. While the jobs are running in the background, you are free to run other programs.

Jobs may also be **suspended**. A suspended job is a job that is not currently running, but is temporarily stopped. After you suspend a job, you can tell the job to continue, in the foreground or the background as needed. Resuming a suspended job will not change the state of the job in any way—the job will continue to run where it left off.

Note that suspending a job is not equal to *interrupting* a job. When you interrupt a running

[2] There are many other processes running on the system as well—"`ps -aux`" lists them all.

process (by hitting your interrupt key, which is usually ctrl-C)[3], it kills the process, for good. Once the job is killed, there's no hope of resuming it; you'll have to re-run the command. Also note that some programs trap the interrupt, so that hitting ctrl-C won't immediately kill the job. This is to allow the program to perform any necessary cleanup operations before exiting. In fact, some programs simply don't allow you to kill them with an interrupt at all.

4.10.3 Backgrounding and killing jobs

Let's begin with a simple example. The command `yes` is a seemingly useless command which sends an endless stream of `y`'s to standard output. (This is actually useful. If you piped the output of `yes` to another command which asked a series of yes and no questions, the stream of `y`'s would confirm all of the questions.)

Try it out.

```
/home/larry# yes
y
y
y
y
y
```

The `y`'s will continue *ad infinitum*. You can kill the process by hitting your interrupt key, which is usually ctrl-C. So that we don't have to put up with the annoying stream of `y`'s, let's redirect the standard output of `yes` to `/dev/null`. As you may remember, `/dev/null` acts as a "black hole" for data. Any data sent to it will disappear. This is a very effective method of quieting an otherwise verbose program.

```
/home/larry# yes > /dev/null
```

Ah, much better. Nothing is printed, but the shell prompt doesn't come back. This is because `yes` is still running, and is sending those inane `y`'s to `/dev/null`. Again, to kill the job, hit the interrupt key.

Let's suppose that we wanted the `yes` command to continue to run, but wanted to get our shell prompt back to work on other things. We can put `yes` into the background, which will allow it to run, but without need for interaction.

One way to put a process in the background is to append an "`&`" character to the end of the command.

```
/home/larry# yes > /dev/null &
[1] 164
/home/larry#
```

[3] The interrupt key can be set using the `stty` command. The default on most systems is ctrl-C, but we can't guarantee the same for your system.

As you can see, we have our shell prompt back. But what is this "`[1] 164`"? And is the `yes` command really running?

The "`[1]`" represents the **job number** for the `yes` process. The shell assigns a job number to every running job. Because `yes` is the one and only job that we're currently running, it is assigned job number `1`. The "`164`" is the process ID, or PID, number given by the system to the job. Either number may be used to refer to the job, as we'll see later.

You now have the `yes` process running in the background, continuously sending a stream of `y`'s to `/dev/null`. To check on the status of this process, use the shell internal command `jobs`.

```
/home/larry# jobs
[1]+  Running                 yes >/dev/null  &
/home/larry#
```

Sure enough, there it is. You could also use the `ps` command as demonstrated above to check on the status of the job.

To terminate the job, use the command `kill`. This command takes either a job number or a process ID number as an argument. This was job number 1, so using the command

```
/home/larry# kill %1
```

will kill the job. When identifying the job with the job number, you must prefix the number with a percent ("`%`") character.

Now that we've killed the job, we can use `jobs` again to check on it:

```
/home/larry# jobs

[1]+  Terminated              yes >/dev/null

/home/larry#
```

The job is in fact dead, and if we use the `jobs` command again nothing should be printed.

You can also kill the job using the process ID (PID) number, which is printed along with the job ID when you start the job. In our example, the process ID is 164, so the command

```
/home/larry# kill 164
```

is equivalent to

```
/home/larry# kill %1
```

You don't need to use the "`%`" when referring to a job by its process ID.

4.10.4 Stopping and restarting jobs

There is another way to put a job into the background. You can start the job normally (in the foreground), **stop** the job, and then restart it in the background. We'll demonstrate this below.

First, start the `yes` process in the foreground, as you normally would:

```
/home/larry# yes > /dev/null
```

Again, because `yes` is running in the foreground, you shouldn't get your shell prompt back.

Now, instead of interrupting the job with ctrl-C, we'll *suspend* the job. Suspending a job doesn't kill it: it only temporarily stops the job until you restart it. To do this, you hit the suspend key, which is usually ctrl-Z.

```
/home/larry# yes > /dev/null
ctrl-Z
[1]+  Stopped                 yes >/dev/null
/home/larry#
```

While the job is suspended, it's simply not running. No CPU time is used for the job. However, you can restart the job, which will cause the job to run again as if nothing ever happened. It will continue to run where it left off.

To restart the job in the foreground, use the command `fg` (for "foreground").

```
/home/larry# fg
yes >/dev/null
```

The shell prints the name of the command again so you're aware of which job you just put into the foreground. Stop the job again, with ctrl-Z. This time, use the command `bg` to put the job into the background. This will cause the command to run just as if you started the command with "`&`" as in the last section.

```
/home/larry# bg
[1]+ yes >/dev/null &
/home/larry#
```

And we have our prompt back. `jobs` should report that `yes` is indeed running, and we can kill the job with `kill` as we did before.

How can we stop the job again? Using ctrl-Z won't work, because the job is in the background. The answer is to put the job in the foreground, with `fg`, and then stop it. As it turns out you can use `fg` on either stopped jobs or jobs in the background.

There is a big difference between a job in the background and a job which is stopped. A stopped job is not running—it's not using any CPU time, and it's not doing any work (the job still occupies system memory, although it may be swapped out to disk). A job in the background is running, and

using memory, as well as completing some task while you do other work. However, a job in the background may try to display text on to your terminal, which can be annoying if you're trying to work on something else. For example, if you used the command

```
/home/larry# yes &
```

without redirecting stdout to `/dev/null`, a stream of `y`'s would be printed to your screen, without any way of interrupting it (you can't use ctrl-C to interrupt jobs in the background). In order to stop the endless `y`'s, you'd have to use the `kill` command (without being able to see what you're typing).

Another note. The `fg` and `bg` commands normally foreground or background the job which was last stopped (indicated by a "`+`" next to the job number when you use the command `jobs`). If you are running multiple jobs at once, you can foreground or background a specific job by giving the job ID as an argument to `fg` or `bg`, as in

```
/home/larry# fg %2
```

(to foreground job number 2), or

```
/home/larry# bg %3
```

(to background job number 3). You can't use process ID numbers with `fg` or `bg`.

Furthermore, using the job number alone, as in

```
/home/larry# %2
```

is equivalent to

```
/home/larry# fg %2
```

Just remember that using job control is a feature of the shell. The commands `fg`, `bg` and `jobs` are internal to the shell. If for some reason you use a shell which does not support job control, don't expect to find these commands available.

In addition, there are some aspects of job control which differ between Bash and Tcsh. In fact, some shells don't provide job control at all—however, most shells available for Linux support job control.

4.11 Using the `vi` Editor

A **text editor** is simply a program used to edit files which contain text, such as a letter, C program, or a system configuration file. While there are many such editors available for Linux, the only editor which you are guaranteed to find on any UNIX system is `vi`— the "visual editor". `vi` is not the

easiest editor to use, nor is it very self-explanatory. However, because it is so common in the UNIX world, and at times you may be required to use it, it deserves some documentation here.

The editor which you prefer to use is mostly a question of personal taste and style. Many users prefer the baroque, self-explanatory and powerful **Emacs**—an editor with more features than any single program in the UNIX world. For example, Emacs has its own built-in dialect of the LISP programming language, and has many extensions (one of which is an "Eliza"-like AI program). However, because Emacs and all of its support files are relatively large, you may not have access to it on many systems. `vi`, on the other hand, is small and powerful, but more difficult to use. However, once you know your way around `vi`, it's actually very easy. It's just the learning curve which is sometimes difficult to cross.

This section is a coherent introduction to `vi`—we won't discuss all of its features, just the ones you need to know to get you started. You can refer to the man page for `vi` if you're interested in learning about more of this editor's features. Or, you can read the book *Learning the* `vi` *Editor* from O'Reilly and Associates. See Appendix B for information.

4.11.1 Concepts

While using `vi`, at any one time you are in one of three modes of operation. These modes are known as *command mode*, *insert mode*, and *last line mode*.

When you start up `vi`, you are in *command mode*. This mode allows you to use certain commands to edit files or to change to other modes. For example, typing "`x`" while in command mode deletes the character underneath the cursor. The arrow keys move the cursor around the file which you're editing. Generally, the commands used in command mode are one or two characters long.

You actually insert or edit text within *insert mode*. When using `vi`, you'll probably spend most of your time within this mode. You start insert mode by using a command such as "`i`" (for "insert") from command mode. While in insert mode, you are inserting text into the document from your current cursor location. To end insert mode and return to command mode, press esc.

Last line mode is a special mode used to give certain extended commands to `vi`. While typing these commands, they appear on the last line of the screen (hence the name). For example, when you type "`:`" from command mode, you jump into last line mode, and can use commands such as "`wq`" (to write the file and quit `vi`), or "`q!`" (to quit `vi` without saving changes). Last line mode is generally used for `vi` commands which are longer than one character.

4.11.2 Starting `vi`

The best way to understand these concepts is to actually fire up `vi` and edit a file. In the example "screens" below, we're only going to show a few lines of text, as if the screen was only six lines high (instead of twenty-five).

The syntax for `vi` is

`vi` ⟨*filename*⟩

where ⟨*filename*⟩ is the name of the file that you wish to edit.

Start up `vi` by typing

```
/home/larry# vi test
```

which will edit the file `test`. You should see something like

```
~
~
~
~
~
~
"test" [New file]
```

The column of "~" characters indicates that you are the end of the file.

4.11.3 Inserting text

You are now in command mode; in order to insert text into the file, press i (which will place you into edit mode), and begin typing.

```
Now is the time for all good men to come to the aid of the party.
~
~
~
~
~
```

While inserting text, you may type as many lines as you wish (pressing return after each, of course), and you may correct mistakes using the backspace key.

To end edit mode, and return to command mode, press esc.

While in command mode, you can use the arrow keys to move around the file. Here, because we only have one line of text, trying to use the up- or down-arrow keys will probably cause `vi` to beep at you.

There are several ways to insert text, other than using the `i` command. For example, the `a` command inserts text beginning *after* the current cursor position, instead of on the current cursor position. For example, use the left arrow key to move the cursor between the words "good" and "men".

```
Now is the time for all good men to come to the aid of the party.
~
~
~
~
~
```

Press [a], to start insert mode, type "wo", and then hit [esc] to return to command mode.

```
Now is the time for all good women to come to the aid of the party.
~
~
~
~
~
```

To begin inserting text at the line below the current one, use the o command. For example, press [o] and type another line or two:

```
Now is the time for all good women to come to the aid of the party.
Afterwards, we'll go out for pizza and beer.
~
~
~
~
```

Just remember that at any time you're either in command mode (where commands such as i, a, or o are valid), or in edit mode (where you're inserting text, followed by [esc] to return to command mode).

4.11.4 Deleting text

From command mode, the x command deletes the character under the cursor. If you press [x] five times, you'll end up with:

```
Now is the time for all good women to come to the aid of the party.
Afterwards, we'll go out for pizza and
~
~
~
~
```

Now press [a], insert some text, followed by [esc]:

```
Now is the time for all good women to come to the aid of the party.
Afterwards, we'll go out for pizza and Diet Coke.
~
~
~
~
```

You can delete entire lines using the command dd (that is, press [d] twice in a row). If your cursor is on the second line, and you type dd,

```
Now is the time for all good women to come to the aid of the party.
~
~
~
~
~
```

To delete the word which the cursor is on, use the `dw` command. Place the cursor on the word "good", and type `dw`.

```
Now is the time for all women to come to the aid of the party.
~
~
~
~
~
```

4.11.5 Changing text

You can replace sections of text using the `R` command. Place the cursor on the first letter in "party", press R, and type the word "hungry".

```
Now is the time for all women to come to the aid of the hungry.
~
~
~
~
~
```

Using `R` to edit text is much like the `i` and `a` commands, but `R` overwrites text, instead of inserting it.

The `r` command replaces the single character under the cursor. For example, move the cursor to the beginning of the word "Now", and type `r` followed by `C`, you'll have:

```
Cow is the time for all women to come to the aid of the hungry.
~
~
~
~
~
```

The "`~`" command changes the case of the letter under the cursor from upper- to lower-case, and vise versa, For example, if you place the cursor on the "o" in "Cow", above, and repeatedly press ~, you'll end up with:

```
COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.
~
~
~
~
~
```

4.11.6 Moving commands

You already know how to use the arrow keys to move around the document. In addition, you can use the `h`, `j`, `k`, and `l` commands to move the cursor left, down, up, and right, respectively. This comes in handy when (for some reason) your arrow keys aren't working correctly.

The `w` command moves the cursor to the beginning of the next word; the `b` moves it to the beginning of the previous word.

The `0` (that's a zero) command moves the cursor to the beginning of the current line, and the `$` command moves it to the end of the line.

When editing large files, you'll want to move forwards or backwards through the file a screenful at a time. Pressing ctrl-F moves the cursor one screenful forward, and ctrl-B moves it a screenful back.

In order to move the cursor to the end of the file, type `G`. You can also move to an arbitrary line; for example, typing the command `10G` would move the cursor to line 10 in the file. To move to the beginning of the file, use `1G`.

You can couple moving commands with other commands, such as deletion. For example, the command `d$` will delete everything from the cursor to the end of the line; `dG` will delete everything from the cursor to the end of the file, and so on.

4.11.7 Saving files and quitting `vi`

To quit `vi` without making changes to the file, use the command `:q!`. When you type the ":", the cursor will move to the last line on the screen; you'll be in last line mode.

```
COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.
~
~
~
~
~
:
```

In last line mode, certain extended commands are available. One of them is `q!`, which quits `vi` without saving. The command `:wq` saves the file and then exits `vi`. The command `ZZ` (from command mode, without the ":") is equivalent to `:wq`.

To save the file without quitting vi, just use `:w`.

4.11.8 Editing another file

To edit another file, use the `:e` command. For example, to stop editing `test`, and edit the file `foo` instead, use the command

```
COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.
~
~
~
~
~
:e foo_
```

If you use `:e` without saving the file first, you'll get the error message

```
No write since last change (":edit!" overrides)
```

which simply means that `vi` doesn't want to edit another file until you save the first one. At this point, you can use `:w` to save the original file, and then use `:e`, or you can use the command

```
COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.
~
~
~
~
~
:e!  foo_
```

The "`!`" tells `vi` that you really mean it—edit the new file without saving changes to the first.

4.11.9 Including other files

If you use the `:r` command, you can include the contents of another file in the current file. For example, the command

```
:r foo.txt
```

would insert the contents of the file `foo.txt` in the text at the current cursor location.

4.11.10 Running shell commands

You can also run shell commands from within `vi`. The `:r!` command works like `:r`, but instead of reading a file, it inserts the output of the given command into the buffer at the current cursor location. For example, if you use the command

```
:r!  ls -F
```

you'll end up with

```
COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE HUNGRY.
letters/
misc/
papers/
~
~
```

You can also "shell out" of `vi`, in other words, run a command from within `vi`, and return to the editor when you're done. For example, if you use the command

```
:!  ls -F
```

the `ls -F` command will be executed, and the results displayed on the screen, but not inserted into the file which you're editing. If you use the command

```
:shell
```

`vi` will start an instance of the shell, allowing you to temporarily put vi "on hold" while you execute other commands. Just logout of the shell (using the `exit` command) to return to `vi`.

4.11.11 Getting help

`vi` doesn't provide much in the way of interactive help (most UNIX programs don't), but you can always read the man page for `vi`. `vi` is a visual front-end to the `ex` editor; it is `ex` which handles many of the last-line mode commands in `vi`. So, in addition to reading the man page for `vi`, see `ex` as well.

4.12 Customizing your Environment

The shell provides many mechanisms to customize your work environment. As we've mentioned before, the shell is more than a command interpreter—it is also a powerful programming language. While writing shell scripts is an extensive subject, we'd like to introduce you to some of the ways that you can simplify your work on a UNIX system by using these advanced features of the shell.

As we have mentioned before, different shells use different syntaxes when executing shell scripts. For example, Tcsh uses a C-like syntax, while Bourne shells use another type of syntax. In this section, we won't be running into many of the differences between the two, but we will assume that shell scripts are executed using the Bourne shell syntax.

4.12.1 Shell scripts

Let's say that you use a series of commands often, and would like to shorten the amount of required typing by grouping all of them together into a single "command". For example, the commands

```
/home/larry# cat chapter1 chapter2 chapter3 > book
/home/larry# wc -l book
/home/larry# lp book
```

would concatenate the files `chapter1`, `chapter2`, and `chapter3` and place the result in the file `book`. Then, a count of the number of lines in `book` would be displayed, and finally `book` would be printed with the `lp` command.

Instead of typing all of these commands, you could group them into a **shell script**. We described shell scripts briefly in Section 4.12.1. The shell script used to run all of these commands would look like

```
#!/bin/sh
# A shell script to create and print the book

cat chapter1 chapter2 chapter3 > book
wc -l book
lp book
```

If this script was saved in the file `makebook`, you could simply use the command

```
/home/larry# makebook
```

to run all of the commands in the script. Shell scripts are just plain text files; you can create them with an editor such as `emacs` or `vi` [4] .

Let's look at this shell script. The first line, "`#!/bin/sh`", identifies the file as a shell script, and tells the shell how to execute the script. It instructs the shell to pass the script to `/bin/sh` for execution, where `/bin/sh` is the shell program itself. Why is this important? On most UNIX systems, `/bin/sh` is a Bourne-type shell, such as Bash. By forcing the shell script to run using `/bin/sh`, we are ensuring that the script will run under a Bourne-syntax shell (instead of, say, a C shell). This will cause your script to run using the Bourne syntax even if you use Tcsh (or another C shell) as your login shell.

The second line is a *comment*. Comments begin with the character "`#`" and continue to the end of the line. Comments are ignored by the shell—they are commonly used to identify the shell script to the programmer.

The rest of the lines in the script are just commands, as you would type them to the shell directly. In effect, the shell reads each line of the script and runs that line as if you had typed it at the shell prompt.

Permissions are important for shell scripts. If you create a shell script, you must make sure that you have execute permission on the script in order to run it[5]. The command

```
/home/larry# chmod u+x makebook
```

can be used to give yourself execute permission on the shell script `makebook`.

[4] `vi` is covered in Section 4.11.

[5] When you create text files, the default permissions usually don't include execute permission.

4.12.2 Shell variables and the environment

The shell allows you to define **variables**, as most programming languages do. A variable is just a piece of data which is given the name.

◇ Note that Tcsh, as well as other C-type shells, use a different mechanism for setting variables than is described here. This discussion assumes the use of a Bourne shell, such as Bash (which you're probably using). See the Tcsh man page for details.

When you assign a value to a variable (using the "`=`" operator), you can access the variable by prepending a "`$`" to the variable name, as demonstrated below.

```
/home/larry# foo="hello there"
```

The variable `foo` is given the value "`hello there`". You can now refer to this value by the variable name, prefixed with a "`$`" character. The command

```
/home/larry# echo $foo
hello there
/home/larry#
```

produces the same results as

```
/home/larry# echo "hello there"
hello there
/home/larry#
```

These variables are internal to the shell. This means that only the shell can access these variables. This can be useful in shell scripts; if you need to keep track of a filename, for example, you can store it in a variable, as above. Using the command `set` will display a list of all defined shell variables.

However, the shell allows you to **export** variables to the **environment**. The environment is the set of variables which all commands that you execute have access to. Once you define a variable inside the shell, exporting it makes that variable part of the environment as well. The `export` command is used to export a variable to the environment.

◇ Again, here we differ between Bash and Tcsh. If you're using Tcsh, another syntax is used for setting environment variables (the `setenv` command is used). See the Tcsh man page for more information.

The environment is very important to the UNIX system. It allows you to configure certain commands just by setting variables which the commands know about.

Here's a quick example. The environment variable `PAGER` is used by the `man` command. It specifies the command to use to display man pages one screenful at a time. If you set `PAGER` to be the name of a command, it will use that command to display the man pages, instead of `more` (which is the default).

Set `PAGER` to "`cat`". This will cause output from `man` to be displayed to the screen all at once, without breaking it up into pages.

```
/home/larry# PAGER="cat"
```

Now, export **PAGER** to the environment.

```
/home/larry# export PAGER
```

Try the command **man ls**. The man page should fly past your screen without pausing for you.

Now, if we set **PAGER** to "**more**", the **more** command will be used to display the man page.

```
/home/larry# PAGER="more"
```

Note that we don't have to use the **export** command after we change the value of **PAGER**. We only need to export a variable once; any changes made to it thereafter will automatically be propagated to the environment.

The man pages for a particular command will tell you if the command uses any environment variables; for example, the **man** man page explains that **PAGER** is used to specify the pager command. Some commands share environment variables; for example, many commands use the **EDITOR** environment variable to specify the default editor to use when one is needed.

The environment is also used to keep track of important information about your login session. An example is the **HOME** environment variable, which contains the name of your home directory.

```
/home/larry/papers# echo $HOME
/home/larry
```

Another interesting environment variable is **PS1**, which defines the main shell prompt. For example,

```
/home/larry# PS1="Your command, please: "
Your command, please:
```

To set the prompt back to our usual (which contains the current working directory followed by a "**#**" symbol),

```
Your command, please:  PS1="\w# "
/home/larry#
```

The **bash** man page describes the syntax used for setting the prompt.

4.12.2.1 The PATH environment variable

When you use the **ls** command, how does the shell know where to find the **ls** executable itself? In fact, **ls** is found in **/bin/ls** on most systems. The shell uses the environment variable **PATH** to locate executable files for commands which you type.

For example, your **PATH** variable may be set to:

```
/bin:/usr/bin:/usr/local/bin:.
```

This is a list of directories for the shell to search, each directory separated by a ":". When you use the command `ls`, the shell first looks for `/bin/ls`, then `/usr/bin/ls`, and so on.

Note that the `PATH` has nothing to do with finding regular files. For example, if you use the command

```
/home/larry# cp foo bar
```

The shell does not use `PATH` to locate the files `foo` and `bar`—those filenames are assumed to be complete. The shell only uses `PATH` to locate the `cp` executable.

This saves you a lot of time; it means that you don't have to remember where all of the command executables are stored. On many systems, executables are scattered about in many places, such as `/usr/bin`, `/bin`, or `/usr/local/bin`. Instead of giving the command's full pathname (such as `/usr/bin/cp`), you can simply set `PATH` to the list of directories that you want the shell to automatically search.

Notice that `PATH` contains ".", which is the current working directory. This allows you to create a shell script or program and run it as a command from your current directory, without having to specify it directly (as in `./makebook`). If a directory isn't on your `PATH`, then the shell will not search it for commands to run—this includes the current directory.

4.12.3 Shell initialization scripts

In addition to shell scripts that you create, there are a number of scripts that the shell itself uses for certain purposes. The most important of these are your **initialization scripts**, scripts automatically executed by the shell when you login.

The initialization scripts themselves are simply shell scripts, as described above. However, they are very useful in setting up your environment by executing commands automatically when you login. For example, if you always use the `mail` command to check your mail when you login, you place the command in your initialization script so it will be executed automatically.

Both Bash and Tcsh distinguish between a **login shell** and other invocations of the shell. A login shell is a shell invoked at login time; usually, it's the only shell which you'll use. However, if you "shell out" of another program, such as `vi`, you start another instance of the shell, which isn't your login shell. In addition, whenever you run a shell script, you automatically start another instance of the shell to execute the script.

The initialization files used by Bash are: `/etc/profile` (set up by the system administrator, executed by all Bash users at login time), `$HOME/.bash_profile` (executed by a login Bash session), and `$HOME/.bashrc` (executed by all non-login instances of Bash). If `.bash_profile` is not present, `.profile` is used instead.

Tcsh uses the following initialization scripts: `/etc/login.csh` (executed by all Tcsh users at login time), `$HOME/.tcshrc` (executed a login time and by all new instances of Tcsh), and `$HOME/.login` (executed at login time, following `.tcshrc`). If `.tcshrc` is not present, `.cshrc` is used instead.

To fully understand the function of these files, you'll need to learn more about the shell itself. Shell programming is a complicated subject, far beyond the scope of this book. See the man pages for `bash` and/or `tcsh` to learn more about customizing your shell environment.

4.13 So You Want to Strike Out on Your Own?

Hopefully we have provided enough information to give you a basic idea of how to use the system. Keep in mind that most of the interesting and important aspects of Linux aren't covered here—these are the very basics. With this foundation, before long you'll be up and running complicated applications and fulfilling the potential of your system. If things don't seem exciting at first, don't despair—there is much to be learned.

One indispensable tool for learning about the system is to read the man pages. While many of the man pages may appear confusing at first, if you dig beneath the surface there is a wealth of information contained therein.

We also suggest reading a complete book on using a UNIX system. There is much more to UNIX than meets the eye—unfortunately, most of it is beyond the scope of this book. Some good UNIX books to look at are listed in Appendix B.

Chapter 5

System Administration

This chapter is an overview to Linux system administration, including a number of advanced features which aren't necessarily for system administrators only. Just as every dog has its day, every system has its administrator, and running the system is a very important and sometimes time-consuming job, even if you're the only user on your system.

We have tried to cover here the most important things about system administration you need to know when you use Linux, in sufficient detail to get you comfortably started. In order to keep it short and sweet, we have only covered the very basics, and have skipped many an important detail. You should read the *Linux System Administrator's Guide* if you are serious about running Linux. It will help you understand better how things work, and how they hang together. At least skim through it so that you know what it contains and know what kind of help you can expect from it.

5.1 About Root, Hats, and the Feeling of Power

As you know, UNIX differentiates between different users, so that what they do to each other and to the system can be regulated (one wouldn't want anybody to be able to read one's love letters, for instance). Each user is given an **account**, which includes a username, home directory, and so on. In addition to accounts given to real people, there are special system-defined accounts which have special privileges. The most important of these is the **root account**, for the username `root`.

5.1.1 The `root` account

Ordinary users are generally restricted so that they can't do harm to anybody else on the system, just to themselves. File permissions on the system are arranged such that normal users aren't allowed to delete or modify files in directories shared by all users (such as `/bin` and `/usr/bin`. Most users also protect their own files with the appropriate file permissions so that other users can't access or modify those files.

There are no such restrictions on `root`. The user `root` can read, modify, or delete any file on

the system, change permissions and ownerships on any file, and run special programs, such as those which partition the drive or create filesystems. The basic idea is that the person or persons who run and take care of the system logs in as `root` whenever it is necessary to perform tasks that cannot be executed as a normal user. Because `root` can do anything, it is easy to make mistakes that have catastrophic consequences when logged in using this account.

For example, as a normal user, if you inadvertently attempt to delete all of the files in `/etc`, the system will not permit you to do so. However, when logged in as `root`, the system won't complain at all. It is very easy to trash your system when using `root`. The best way to prevent accidents is to:

- Sit on your hands before you press `return` on a command which may cause damage. For example, if you're about to clean out a directory, before hitting `return`, re-read the entire command and make sure that it is correct.
- Don't get accustomed to using `root`. The more comfortable you are in the role of the `root` user, the more likely you are to confuse your privileges with those of a normal user. For example, you might *think* that you're logged in as `larry`, when you're really logged in as `root`.
- Use a different prompt for the `root` account. You should change `root`'s `.bashrc` or `.login` file to set the shell prompt to something other than your regular user prompt. For example, many people use the character "`$`" in prompts for regular users, and reserve the character "`#`" for the `root` user prompt.
- Only login as `root` when absolutely necessary. And, as soon as you're finished with your work as `root`, log out. The less you use the `root` account, the less likely you'll be to do damage on your system.

Of course, there is a breed of UNIX hackers out there who use `root` for virtually everything. But every one of them has, at some point, made a silly mistake as `root` and trashed the system. The general rule is, until you're familiar with the lack of restrictions on `root`, and are comfortable using the system without such restrictions, login as `root` sparingly.

Of course, everyone makes mistakes. Linus Torvalds himself once accidentally deleted the entire kernel directory tree on his system. Hours of work were lost forever. Fortunately, however, because of his knowledge of the filesystem code, he was able to reboot the system and reconstruct the directory tree by hand on disk.

Put another way, if you picture using the `root` account as wearing a special magic hat that gives you lots of power, so that you can, by waving your hand, destroy entire cities, it is a good idea to be a bit careful about what you do with your hands. Since it is easy to move your hand in a destructive way by accident, it is not a good idea to wear the magic hat when it is not needed, despite the wonderful feeling.

5.1.2 Abusing the system

Along with the feeling of power comes the tendency to do harm. This is one of the grey areas of UNIX system administration, but everyone goes through it at some point in time. Most users of

UNIX systems never have the ability to wield this power—on university and business UNIX systems, only the highly-paid and highly-qualified system administrators ever login is `root`. In fact, at many such institutions, the `root` password is a highly guarded secret: it is treated as the Holy Grail of the institution. A large amount of hubbub is made about logging in as `root`; it is portrayed as a wise and fearsome power, given only to an exclusive cabal.

This kind of attitude towards the `root` account is, quite simply, the kind of thing which breeds malice and contempt. Because `root` is so fluffed-up, when some users have their first opportunity to login as `root` (either on a Linux system or elsewhere), the tendency is to use `root`'s privileges in a harmful manner. I have known so-called "system administrators" who read other user's mail, delete user's files without warning, and generally behave like children when given such a powerful "toy".

Because `root` has such privilege on the system, it takes a great deal of maturity and self-control to use the account as it was intended—to run the system. There is an unspoken code of honor which exists between the system administrator and the users on the system. How would you feel if your system administrator was reading your e-mail or looking over your files? There is still no strong legal precedent for electronic privacy on time-sharing computer systems. On UNIX systems, the `root` user has the ability to forego all security and privacy mechanisms on the system. It is important that the system administrator develop a trusting relationship with the users on the system. I can't stress that enough.

5.1.3 Dealing with users

UNIX security is rather lax by design. Security on the system was an afterthought—the system was originally developed in an environment where users intruding upon other users was simply unheard of. Because of this, even with security measures, there is still the ability for normal users to do harm.

System administrators can take two stances when dealing with abusive users: they can be either paranoid or trusting. The paranoid system administrator usually causes more harm than he or she prevents. One of my favorite sayings is, "Never attribute to malice anything which can be attributed to stupidity." Put another way, most users don't have the ability or knowledge to do real harm on the system. 90% of the time, when a user is causing trouble on the system (by, for instance, filling up the user partition with large files, or running multiple instances of a large program), the user is simply unaware that what he or she is doing is a problem. I have come down on users who were causing a great deal of trouble, but they were simply acting out of ignorance—not malice.

When you deal with users who are causing potential trouble, don't be accusative. The old rule of "innocent until proven guilty" still holds. It is best to simply talk to the user, and question about the trouble, instead of causing a confrontation. The last thing you want to do is be on the user's bad side. This will raise a lot of suspicion about you—the system administrator—running the system correctly. If a user believes that you distrust or dislike them, they might accuse you of deleting files or breaching privacy on the system. This is certainly not the kind of position that you want to be in.

If you do find that a user has been attempting to "crack" the system, or was intentionally doing harm to the system, don't return the malicious behavior with malice of your own. Instead, simply

provide a warning—but be flexible. In many cases, you may catch a user "in the act" of doing harm to the system—give them a warning. Tell them not to let it happen again. However, if you *do* catch them causing harm again, be absolutely sure that it is intentional. I can't even begin to describe the number of cases where it appeared as though a user was causing trouble, when in fact it was either an accident or a fault of my own.

5.1.4 Setting the rules

The best way to run a system is not with an iron fist. That may be how you run the military, but UNIX was not designed for such discipline. It makes sense to lay down a simple and flexible set of guidelines for users—but remember, the fewer rules you have, the less chance there is of breaking them. Even if your rules for using the system are perfectly reasonable and clear, users will always at times break these rules without intending to. This is especially true in the case of new UNIX users, who are just learning the ropes of the system. It's not patently obvious, for example, that you shouldn't download a gigabyte of files and mail them to everyone on the system. Users need help understanding the rules, and why they are there.

If you do specify usage guidelines for your system, make sure that the reason behind a particular guideline is made clear. If you don't, then users will find all sorts of creative ways to get around the rule, and not know that they are in fact breaking it.

5.1.5 What it all means

We can't tell you how to run your system to the last detail. Most of the philosophy depends on how you're using the system. If you have many users, things are much different than if you only have a few users, or if you're the only user on the system. However, it's always a good idea—in any situation—to understand what being the system administrator really means.

Being the system administrator doesn't make you a UNIX wizard. There are many system admins out there who know very little about UNIX. Likewise, there are many "normal" users out there who know more about UNIX than any system administrator could. Also, being the system administrator does not allow you to use malice against your users. Just because the system gives you the privilege to mess with user files does not mean that you have any right to do so.

Lastly, being the system administrator is really not a big deal. It doesn't matter if your system is a little 386 or a Cray supercomputer. Running the system is the same, regardless. Knowing the `root` password isn't going to get you money, fame, or girls. It will allow you to maintain the system, and keep it running. That's it.

5.2 Booting the System

There are several ways to boot the system, either from floppy or from the hard drive.

5.2.1 Using a boot floppy

Many people boot Linux using a "boot floppy" which contains a copy of the Linux kernel. This kernel has the Linux root partition coded into it, so it will know where to look on the hard drive for the root filesystem. (The `rdev` command can be used to set the root partition in the kernel image; see below.) This is the type of floppy created by SLS during installation, for example.

To create your own boot floppy, first locate the kernel image on your hard disk. It should be in the file `/Image` or `/etc/Image`. Some installations use the file `/vmlinux` for the kernel.

You may instead have a compressed kernel. A compressed kernel uncompresses itself into memory at boot time, and takes up much less space on the hard drive. If you have a compressed kernel, it may be found in the file `/zImage` or `/etc/zImage`.

Once you know where the kernel is, set the root device in the kernel image to the name of your Linux root partition with the `rdev` command. The format of the command is

`rdev` ⟨*kernel-name*⟩ ⟨*root-device*⟩

where ⟨*kernel-name*⟩ is the name of the kernel image, and ⟨*root-device*⟩ is the name of the Linux root partition. For example, to set the root device in the kernel `/etc/Image` to `/dev/hda2`, use the command

```
# rdev /etc/Image /dev/hda2
```

`rdev` can set other options in the kernel as well, such as the default SVGA mode to use at boot time. Just use "`rdev`" with no arguments to get a help message.

After setting the root device, you can simply copy the kernel image to the floppy. Whenever copying data a floppy, it's a good idea to MS-DOS format the floppy first. This lays down the sector and track information on the floppy, so it can be detected as either high or low density.

For example, to copy the kernel in the file `/etc/Image` to the floppy in `/etc/fd0`, use the command

```
# cp /etc/Image /dev/fd0
```

This floppy should now boot Linux.

5.2.2 Using LILO

Another method of booting is to use LILO, a program which resides in the boot sector of your hard disk. This program is executed when the system is booted from the hard disk, and can automatically boot up Linux from a kernel image stored on the hard drive itself.

LILO can also be used as a first-stage boot loader for several operating systems, allowing you to select at boot time which operating system (such as Linux or MS-DOS) to boot. When you boot using LILO, the default operating system is booted unless you press ctrl, alt, or shift during the

bootup sequence. If you press any of these keys, you will be provided with a boot prompt, at which you type the name of the operating system to boot (such as "`linux`" or "`msdos`"). If you press tab at the boot prompt, a listing of available operating systems will be provided.

LILO is located in the directory `/etc/lilo` (if you have it). The easy way to install LILO is to edit the configuration file, `/etc/lilo/config`, and then run the command

```
# /etc/lilo/lilo
```

The LILO configuration file contains a "stanza" for each operating system that you want to boot. The best way to demonstrate this is with an example LILO config file. The below setup is for a system which has a Linux root partition on `/dev/hda1`, and an MS-DOS partition on `/dev/hda2`.

```
# Tell LILO to modify the boot record on /dev/hda (the first
# non-SCSI hard drive). If you boot from a drive other than /dev/hda,
# change the following line.
boot = /dev/hda

# Name of the boot loader. No reason to modify this unless you're doing
# some serious hacking on LILO.
install = /etc/lilo/boot.b

# Have LILO perform some optimization.
compact

# Stanza for Linux root partition on /dev/hda1.
image = /etc/Image   # Location of kernel
   label = linux     # Name of OS (for the LILO boot menu)
   root = /dev/hda1  # Location of root partition
   vga = ask         # Tell kernel to ask for SVGA modes at boot time

# Stanza for MSDOS partition on /dev/hda2.
other = /dev/hda2    # Location of partition
   table = /dev/hda  # Location of partition table for /dev/hda2
   label = msdos     # Name of OS (for boot menu)
```

The first operating system stanza in the config file will be the default OS for LILO to boot. You can select another OS to boot at the LILO boot prompt, as discussed above.

The program `/etc/lilo/QuickInst` will ask you questions about your setup and generate a LILO config file for you.

Remember that every time you update the kernel image on disk, you should rerun `/etc/lilo/lilo` in order for the changes to be reflected on the boot sector of your drive.

Also note that if you use the "`root =`" line, above, there's no reason to use `rdev` to set the root partition in the kernel image. LILO sets it for you at boot time.

The Linux FAQ (see Appendix B) provides more information on LILO, including how to use LILO to boot with OS/2's Boot Manager.

5.3 Shutting Down

Shutting down a Linux system is a bit tricky. Remember that you should never just turn off the power or hit the reset switch while the system is running. The kernel keeps track of disk I/O in memory buffers. If you reboot the system without giving the kernel the chance to write its buffers to disk, you can corrupt your filesystems.

Other precautions are taken at shutdown time as well. All processes are sent a signal, which allows them to die gracefully (writing and closing all files, and so on). Filesystems are unmounted for safety. If you wish, the system can also alert users that the system is going down and give them a change to log off.

The easiest way to shutdown is with the **shutdown** command. The format of the command is

```
shutdown ⟨time⟩ ⟨warning-message⟩
```

The ⟨*time*⟩ argument is the time to shutdown the system (in the format *hh:mm:ss*), and ⟨*warning-message*⟩ is a message displayed on all user's terminals before shutdown. Alternately, you can specify the ⟨*time*⟩ as "**now**", to shutdown immediately. The **-r** option may be given to **shutdown** to reboot the system after shutting down.

For example, to shutdown the system at 8:00pm, use the command

```
# shutdown -r 20:00
```

The command **halt** may be used to force an immediate shutdown, without any warning messages or grace period. **halt** is useful if you're the only one using the system, and want to shut down the system and turn it off.

◇ Don't turn off the power or reboot the system until you see the message:

```
The system is halted
```

It is very important that you shutdown the system "cleanly" using the **shutdown** or **halt** commands. On some systems, pressing ctrl-alt-del will be trapped and cause a **shutdown**; on other systems, however, using the "Vulcan nerve pinch" will reboot the system immediately and may cause disaster.

5.4 Managing Users

Whether or not you have many users on your system, it's important to understand the aspects of user management under Linux. Even if you're the only user, you should presumably have a separate account for yourself (an account other than **root** to do most of your work).

Each person using the system should have his or her own account. It is seldom a good idea to have several people share the same account. Not only is security an issue, but accounts are used to uniquely identify users to the system. You need to be able to keep track of who is doing what.

5.4.1 User management concepts

The system keeps track of a number of pieces of information about each user. They are summarized below.

username The username is the unique identifier given to every user on the system. Examples of usernames are `larry`, `karl`, and `mdw`. Letters and digits may be used, as well as the characters "_" (underscore) and "." (period). Usernames are usually limited to 8 characters in length.

user ID The user ID, or UID, is a unique number given to every user on the system. The system usually keeps track of information by UID, not username.

group ID The group ID, or GID, is the ID of the user's default group. In Section 4.9 we discussed group permissions; each user belongs to one or more groups defined by the system administrator. More about this below.

password The system also stores the user's encrypted password. The `passwd` command is used to set and change user passwords.

full name The user's "real name" or "full name" is stored along with the username. For example, the user `schmoj` may have the name "Joe Schmo" in real life.

home directory The home directory is the directory in which the user is initially placed at login time. Every user should have his or her own home directory, usually found under `/home`.

login shell The user's login shell is the shell which is started for the user at login time. Examples are `/bin/bash` and `/bin/tcsh`.

The file `/etc/passwd` contains this information about users. Each line in the file contains information about a single user; The format of each line is

```
username:encrypted password:UID:GID:full name:home directory:login shell
```

An example might be:

```
kiwi:Xv8Q981g71oKK:102:100:Laura Poole:/home/kiwi:/bin/bash
```

As we can see, the first field, "`kiwi`", is the username.

The next field, "`Xv8Q981g71oKK`", is the encrypted password. Passwords are not stored on the system in any human-readable format. The password is encrypted using itself as the secret key. In other words, you need to know the password to decrypt it. This form of encryption is fairly secure.

Some systems use "shadow password" in which password information is relegated to the file `/etc/shadow`. Because `/etc/passwd` is world-readable, `/etc/shadow` provides some degree of extra

security because it is not. Shadow password provides some other features such as password expiration and so on; we will not go into these features here.

The third field, "`102`", is the UID. This must be unique for each user. The fourth field, "`100`", is the GID. This user belongs to the group numbered 100. Group information, like user information, is stored in the file `/etc/group`. See Section 5.4.5 for more information.

The fifth field is the user's full name, "`Laura Poole`". The last two fields are the user's home directory (`/home/kiwi`) and login shell (`/bin/bash`), respectively. It is not required that the user's home directory be given the same name as the username. It does help identify the directory, however.

5.4.2 Adding users

When adding a user, there are several steps to be taken. First, the user must be given an entry in `/etc/passwd`, with a unique username and UID. The GID, fullname, and other information must be specified. The user's home directory must be created, and the permissions on the directory set so that the user owns the directory. Shell initialization files must be provided in the new home directory and other system-wide configuration must be done (for example, setting up a spool for incoming e-mail for the new user).

While it is not difficult to add users by hand (I do), when you are running a system with many users it is easy to forget something. The easiest way to add users is to use an interactive program which asks you for the required information and updates all of the system files automatically. The name of this program is `useradd` or `adduser`, depending on what software was installed. The man pages for these commands should be fairly self-explanatory.

5.4.3 Deleting users

Similarly, deleting users can be accomplished with the commands `userdel` or `deluser` depending on what software was installed on the system.

If you'd like to temporarily "disable" a user from logging into the system (without deleting the user's account), you can simply prepend an asterisk ("`*`") to the password field in `/etc/passwd`. For example, changing `kiwi`'s `/etc/passwd` entry to

```
kiwi:*Xv8Q981g71oKK:102:100:Laura Poole:/home/kiwi:/bin/bash
```

will restrict `kiwi` from logging in.

5.4.4 Setting user attributes

After you have created a user, you may need to change attributes for that user, such as home directory or password. The easiest way to do this is to change the values directly in `/etc/passwd`. To set a user's password, use the `passwd` command. For example,

```
# passwd larry
```

will change `larry`'s password. Only `root` may change other user's password in this manner. Users can change their own passwords with `passwd` as well.

On some systems, the commands `chfn` and `chsh` will be available to allow users to set their own fullname and login shell attributes. If not, they will have to ask the system administrator to change these attributes for them.

5.4.5 Groups

As we have mentioned, each user belongs to one or more groups. The only real importance of group relationships pertains to file permissions, as you'll recall from Section 4.9, each file has a "group ownership" and a set of group permissions which defines how users in that group may access the file.

There are several system-defined groups such as `bin`, `mail`, and `sys`. Users should not belong to any of these groups; they are used for system file permissions. Instead, users should belong to an individual group such as `users`. If you want to be cute, you can maintain several groups of users such as `student`, `staff`, and `faculty`.

The file `/etc/group` contains information about groups. The format of each line is

```
group name:password:GID:other members
```

Some example groups might be:

```
root:*:0:
users:*:100:mdw,larry
guest:*:200:
other:*:250:kiwi
```

The first group, `root`, is a special system group reserved for the `root` account. The next group, `users`, is for regular users. It has a GID of 100. The users `mdw` and `larry` are given access to this group. Remember that in `/etc/passwd` each user was given a default GID. However, users may belong to more than one group, by adding their usernames to other group lines in `/etc/group`. The `groups` command lists what groups you are given access to.

The third group, `guest`, is for guest users, and `other` is for "other" users. The user `kiwi` is given access to this group as well.

As you can see, the "password" field of `/etc/group` is rarely used. It is sometimes used to set a password on group access. This is seldom necessary. To protect users from changing into priveleged groups (with the `newgroup` command), set the password field to "`*`".

The commands `addgroup` or `groupadd` may be used to add groups to your system. Usually, it's easier just to add entries in `/etc/group` yourself, as no other configuration needs to be done to add a group. To delete a group, simply delete its entry in `/etc/group`.

5.5 Archiving and Compressing Files

Before we can talk about backups, we need to introduce the tools used to archive software on UNIX systems.

5.5.1 Using `tar`

The `tar` command is most often used to archive software.

The format of the `tar` command is

`tar` ⟨*options*⟩ ⟨*file1*⟩ ⟨*file2*⟩ ... ⟨*fileN*⟩

where ⟨*options*⟩ is the list of commands and options for `tar`, and ⟨*file1*⟩ through ⟨*fileN*⟩ is the list of files to add or extract from the archive.

For example, the command

```
# tar cvf backup.tar /etc
```

would pack all of the files in `/etc` into the tar archive `backup.tar`. The first argument to `tar`—"`cvf`"—is the `tar` "command". "`c`" tells `tar` to create a new archive file. The "`v`" option forces `tar` into verbose mode—printing each filename as it is archived. The "`f`" option tells `tar` that the next argument—`backup.tar`—is the name of the archive to create. The rest of the arguments to `tar` are the file and directory names to add to the archive.

The command

```
# tar xvf backup.tar
```

will extract the tar file `backup.tar` in the current directory. This can sometimes be dangerous—when extracting files from a tar file, old files are overwritten.

Furthermore, before extracting tar files it is important to know where the files should be unpacked. For example, let's say you archived the following files: `/etc/hosts`, `/etc/group`, and `/etc/passwd`. If you use the command

```
# tar cvf backup.tar /etc/hosts /etc/group /etc/passwd
```

the directory name `/etc/` is added to the beginning of each filename. In order to extract the files to the correct location, you would need to use the following commands:

```
# cd /
# tar xvf backup.tar
```

because files are extracted with the pathname saved in the archive file.

If, however, you archived the files with the command

```
# cd /etc
# tar cvf hosts group passwd
```

the directory name is not saved in the archive file. Therefore, you would need to "`cd /etc`" before extracting the files. As you can see, how the tar file is created makes a large difference in where you extract it. The command

```
# tar tvf backup.tar
```

may be used to display an "index" of the tar file before unpacking it. In this way you can see what directory the filenames in the archive are stored relative to, and can extract the archive from the correct location.

5.5.2 `gzip` and `compress`

Unlike archiving programs for MS-DOS, `tar` does not automatically compress files as it archives them. Therefore, if you are archiving two 1-megabyte files, the resulting tar file will be two megabytes in size. The `gzip` command may be used to compress a file (the file to compress need not be a tar file). The command

```
# gzip -9 backup.tar
```

will compress `backup.tar` and leave you with `backup.tar.gz`, the compressed version of the file. The `-9` switch tells `gzip` to use the highest compression factor.

The `gunzip` command may be used to uncompress a gzipped file. Equivalently, you may use "`gzip -d`".

`gzip` is a relatively new tool in the UNIX community. For many years, the `compress` command was used instead. However, because of several factors[1], `compress` is being phased out.

`compress`ed files end in the extension `.Z`. For example, `backup.tar.Z` is the compressed version of `backup.tar`, while `backup.tar.gz` is the gzipped version[2]. The `uncompress` command is used to expand a `compress`ed file; `gunzip` knows how to handle `compress`ed files as well.

5.5.3 Putting them together

Therefore, to archive a group of files and compress the result, you can use the commands:

```
# tar cvf backup.tar /etc
# gzip -9 backup.tar
```

The result will be `backup.tar.gz`. To unpack this file, use the reverse set of commands:

[1] These factors include a software patent dispute against the `compress` algorithm and the fact that `gzip` is much more efficient than `compress`.

[2] To add further confusion, for some time the extension `.z` (lowercase "z") was used for gzipped files. The official `gzip` extension is now `.gz`.

```
# gunzip backup.tar.gz
# tar xvf backup.tar
```

Of course always make sure that you are in the correct directory before unpacking a tar file.

You can use some UNIX cleverness to do all of this on one command line, as in the following:

```
# tar cvf - /etc | gzip -9c ¿ backup.tar.gz
```

Here, we are sending the tar file to "-", which stands for `tar`'s standard output. This is piped to `gzip`, which compresses the incoming tar file, and the result is saved in `backup.tar.gz`.

The `-c` option to `gzip` tells `gzip` to send its output to stdout, which is redirected to `backup.tar.gz`.

A single command used to unpack this archive would be:

```
# gunzip -c backup.tar.gz | tar xvf -
```

Again, `gunzip` uncompresses the contents of `backup.tar.gz` and sends the resulting tar file to stdout. This is piped to `tar`, which reads "-", this time referring to `tar`'s standard input.

Happily, the `tar` command also includes the `-z` option to automatically compress/uncompress files on the fly. However, it does this as per the `compress` algorithm—it does not use `gzip`. (Please note that the newest versions of GNU `tar` do in fact use `gzip` when using the `-z` option. However, if you use a `tar` binary from the Stone Age, like me, then don't expect `-z` to use `gzip` compression.)

For example, the command

```
# tar cvfz backup.tar.Z /etc
```

is equivalent to

```
# tar cvf backup.tar /etc
# compress backup.tar
```

Just as the command

```
# tar xvfz backup.tar.Z
```

may be used instead of

```
# uncompress backup.tar.Z
# tar xvf backup.tar
```

Refer to the man pages for `tar` and `gzip` for more information.

5.6 Using Floppies and Making Backups

Floppies are usually used as backup media. If you don't have a tape drive connected to your system, floppy disks can be used (although they are slower and somewhat less reliable).

You may also use floppies to hold individual filesystems—in this way, you can **mount** the floppy to access the data on it.

5.6.1 Using floppies for backups

The easiest way to make a backup using floppies is with `tar`. The command

```
# tar cvfzM /dev/fd0 /
```

will make a complete backup of your system using the floppy drive `/dev/fd0`. The "`M`" option to `tar` allows the backup to be a multivolume backup; that is, when one floppy is full, `tar` will prompt for the next. The command

```
# tar xvfzM /dev/fd0
```

can be used to restore the complete backup. This method can also be used if you have a tape drive (`/dev/rmt0`) connected to your system.

Note that using the method, you must settle for the `compress` algorithm; `tar` doesn't use `gzip` with the "`z`" option. Several other programs exist for making multiple-volume backups; the `backflops` program on `tsx-11.mit.edu` may come in handy.

Making a complete backup of the system can be time- and resource-consuming. Most system administrators use a incremental backup policy, in which every month a complete backup is taken, and every week only those files which have been modified in the last week are backed up. In this case, if you trash your system in the middle of the month, you can simply restore the last full monthly backup, and then restore the last weekly backups as needed.

The `find` command can be useful in locating files which have changed since a certain date. Several scripts for managing incremental backups can be found on `sunsite.unc.edu`.

5.6.2 Using floppies as filesystems

You can create a filesystem on a floppy just as you would on a hard drive partition. For example,

```
# mke2fs /dev/fd0 1440
```

creates a filesystem on the floppy in `/dev/fd0`. The size of the filesystem must correspond to the size of the floppy. High-density 3.5" disks are 1.44 megabytes, or 1440 blocks, in size. High-density 5.25" disks are 1200 blocks.

In order to access the floppy, you must **mount** the filesystem contained on it. The command

```
# mount -t ext2 /dev/fd0 /mnt
```

will mount the floppy in `/dev/fd0` on the directory `/mnt`. Now, all of the files on the floppy will appear under `/mnt` on your drive. The "`-t ext2`" specifies an ext2fs filesystem type. If you created another type of filesystem on the floppy, you'll need to specify its type to the `mount` command.

The "mount point" (the directory where you're mounting the filesystem) needs to exist when you use the `mount` command. If it doesn't exist, simply create it with `mkdir`.

See Section 5.8 for more information on filesystems, mounting, and mount points.

◇ Note that any I/O to the floppy is buffered just as hard disk I/O is. If you change data on the floppy, you may not see the drive light come on until the kernel flushes its I/O buffers. It's important that you not remove a floppy before you unmount it; this can be done with the command

```
# umount /dev/fd0
```

Do not simply switch floppies as you would on an MS-DOS system; whenever you change floppies, `umount` the first one and `mount` the next.

5.7 Upgrading and Installing New Software

Another duty of the system administrator is upgrading and installing new software.

The Linux community is very dynamic. New kernel releases come out every few weeks, and other software is updated almost as often. Because of this, new Linux users often feel the need to upgrade their systems constantly to keep up the the rapidly changing pace. Not only is this unnecessary, it's a waste of time: to keep up with all of the changes in the Linux world, you would be spending all of your time upgrading and none of your time using the system.

So, when should you upgrade? Some people feel that you should upgrade when a new distribution release is made—for example, when SLS comes out with a new version. Many Linux users completely reinstall their system with the newest SLS release every time. This, also, is a waste of time. In general, changes to SLS releases are small. Downloading and reinstalling 30 disks when only 10% of the software has been actually modified is, of course, pointless.

The best way to upgrade your system is to do it by hand: only upgrade those software packages which you know that you should upgrade. This scares a lot of people: they want to know what to upgrade, and how, and what will break if they don't upgrade. In order to be successful with Linux, it's important to overcome your fears of "doing it yourself"— which is what Linux is all about. In fact, once you have your system working and all software correctly configured, reinstalling with the newest SLS release will no doubt wipe all of your configuration and things will be broken again, just as they were when you first installed your system. Setting yourself back in this manner is unnecessary—all that is needed is some know-how about upgrading your system, and how to do it right.

You'll find that when you upgrade one component of your system that other things should not break. For example, most of the software on my system is left over from an ancient 0.96 MCC

Interim installation. Yet, I run the newest version of the kernel and libraries with this software with no problem. For the most part, senselessly upgrading to "keep up with the trend" is not important at all. This isn't MS-DOS or Microsoft Windows. There is no important reason to run the newest version of all of the software. If you find that you would like or need features in a new version, then upgrade. If not, then don't. In other words, only upgrade what you have to, and when you have to. Don't just upgrade for the sake of upgrading. That will waste a lot of time and effort trying to keep up.

The most important software to upgrade on your system is the kernel, the libraries, and the `gcc` compiler. These are the three essential parts of your system, and in some cases they all depend on each other for everything to work successfully. Most of the other software on your system does not need to be upgraded periodically.

5.7.1 Upgrading the kernel

Upgrading the kernel is simply a matter of getting the sources and compiling them yourself. You must compile the kernel yourself in order to enable or disable certain features, as well as to ensure that the kernel will be optimized to run on your machine. The process is quite painless.

The kernel sources may be retrieved from any of the Linux FTP sites (see Section C for a list). On `sunsite.unc.edu`, for instance, the kernel sources are found in `/pub/Linux/kernel`. Kernel versions are numbered using a version number and a patchlevel. For example, kernel version 0.99 patchlevel 11 is usually written as `0.99.pl11`, or just `0.99.11`.

The kernel sources are released as a gzipped tar file[3]. For example, the file containing the 0.99.pl11 kernel sources is `linux-0.99.11.tar.gz`.

Unpack this tar file from the directory `/usr/src`; it creates the directory `/usr/src/linux` which contains the kernel sources. You should delete or rename your existing `/usr/src/linux` before unpacking the new version.

Once the sources are unpacked, you need to make sure that two symbolic links in `/usr/include` are correct. To create these links, use the commands

```
# ln -sf /usr/src/linux/include/linux /usr/include/linux
# ln -sf /usr/src/linux/include/asm /usr/include/asm
```

Once you have created these links once, there is no reason to create them again when you install the next version of the kernel sources. (See Section 5.9.3 for more about symbolic links.)

Note that in order to compile the kernel, you must have the `gcc` and `g++` C and C++ compilers installed on your system. You may need to have the most recent versions of these compilers: see Section 5.7.3, below, for more information.

To compile the kernel, first `cd` to `/usr/src/linux`. Run the command `make config`. This

[3] Often, a patch file is also released for the current kernel version which allows you to patch your current kernel sources from the last patchlevel to the current one (using the program `patch`). In most cases, however, it's usually easier to install the entire new version of the kernel sources

command will prompt you for a number of configuration options, such as what filesystem types you wish to include in the new kernel.

Next, edit `/usr/src/linux/Makefile`. Be sure that the definition for `ROOT_DEV` is correct—it defines the device uses as the root filesystem at boot time. The usual definition is

```
ROOT_DEV = CURRENT
```

Unless you are changing your root filesystem device, there is no reason to change this.

Next, run the command `make dep` to fix all of the source dependencies. This is a very important step.

Finally, you're ready to compile the kernel. The command `make Image` will compile the kernel and leave the new kernel image in the file `/usr/src/linux/Image`. Alternately, the command `make zImage` will compile a compressed kernel image, which uncompresses itself at boot time and uses less drive space.

Once you have the kernel compiled, you need to either copy it to a boot floppy (with a command such as "`cp Image /dev/fd0`") or install it using LILO to boot from your hard drive. See Section 5.2.2 for more information.

5.7.2 Upgrading the libraries

As mentioned before, most of the software on the system is compiled to use shared libraries, which contain common subroutines shared among different programs.

If you see the message

```
Incompatible library version
```

when attempting to run a program, then you need to upgrade to the version of the libraries which the program requires. Libraries are back-compatible; that is, a program compiled to use an older version of the libraries should work with the new version of the libraries installed. However, the reverse is not true.

The newest version of the libraries can be found on the Linux FTP sites. On `sunsite.unc.edu`, they are located in `/pub/Linux/GCC`. The "release" files there should explain what files you need to download and how to install them. Briefly, you should get the files `image-`*version*`.tar.gz` and `inc-`*version*`.tar.gz` where `version` is the version of the libraries to install, such as `4.4.1`. These are gzipped tar files; the `image` file contains the library images to install in `/lib` and `/usr/lib`. The `inc` file contains include files to install in `/usr/include`

The `release-`*version*`.tar.gz` should explain the installation procedure in detail (the exact instructions vary for each release). In general you need to install the library `.a` and `.sa` files in `/usr/lib`. These are the libraries used at compilation time.

In addition, the shared library image files, `libc.so.`*version* are installed in `/lib`. These are the shared library images loaded at runtime by programs using the libraries. Each library has a symbolic link using the major version number of the library in `/lib`.

For example, the `libc` library version 4.4.1 has a major version number of `4`. The file containing the library is `libc.so.4.4.1`. A symbolic link of the name `libc.so.4` is also in `/lib` pointing to this file. You need to change this symbolic link when upgrading the libraries. For example, when upgrading from `libc.so.4.4` to `libc.so.4.4.1`, you need to change the symbolic link to point to the new version.

◇ It is very important that you change the symbolic link in one step, as given below. If you somehow delete the symbolic link `libc.so.4`, then programs which depend on the link (including basic utilities like `ls` and `cat`) will stop working. Use the following command to update the symbolic link `libc.so.4` to point to the file `libc.so.4.4.1`:

```
# ln -sf /lib/libc.so.4.4.1 /lib/libc.so.4
```

You also need to change the symbolic link `libm.so.`*version* in the same manner. If you are upgrading to a different version of the libraries substitute to appropriate filenames above. The library release notice should explain the details. (See Section 5.9.3 for more information about symbolic links.)

5.7.3 Upgrading gcc

The `gcc` C and C++ compiler is used to compile software on your system, most importantly the kernel. The newest version of `gcc` is found on the Linux FTP sites. On `sunsite.unc.edu`, it is found in the directory `/pub/Linux/GCC` (along with the libraries). There should be a `release` file for the `gcc` distribution detailing what files you need to download and how to install them.

5.7.4 Upgrading other software

Upgrading other software is usually just a matter of downloading the appropriate files and installing them. Most software for Linux is distributed at gzipped tar files, including either sources or binaries or both. If binaries are not included in the release, you may need to compile them yourself; usually, this means typing `make` in the directory where the sources are held.

Reading the USENET newsgroup `comp.os.linux.announce` for announcements of new software releases is the easiest way to find out about new software. Whenever you are looking for software on an FTP site, downloading the `ls-lR` index file from the FTP site and using `grep` to find the files in question is the easiest way to locate software. If you have `archie` available to you, it can be of assistance as well[4]. See Appendix B for more details.

One handy source of Linux software is the SLS distribution disk images. Each disk contains a number of `.tgz` files which are simply gzipped tar files. Instead of downloading the disks, you can download the desired `.tgz` files from the SLS directories on the FTP site and install them directly. If you run the SLS distribution, the `sysinstall` command can be used to automatically load and install a complete series of disks. For example, the command

```
# sysinstall -series t
```

[4] If you don't have `archie`, you can telnet to an `archie` server such as `archie.rutgers.edu`, login as "`archie`" and use the command "`help`"

will install the entire SLS `t` series of disks. Of course, most of the time you may not download or wish to install an entire series, in which case you'll need to unpack the `.tgz` files by hand.

Again, it's usually not a good idea to upgrade by reinstalling with the newest version of SLS, or another distribution. SLS in particular was not designed to be upgradeable. If you reinstall in this way, you will no doubt wreck your current installation, including user directories and all of your customized configuration. The best way to upgrade software is piecewise; that is, if there is a program that you use often that has a new version, upgrade it. Otherwise, don't bother. Rule of thumb: If it ain't broke, don't fix it. If your current software works, there's no reason to upgrade.

5.8 Managing Filesystems

Another task of the system administrator is taking care of filesystems. Most of this job entails periodically checking the filesystems for damage or corrupted files; many systems automatically check the filesystems at boot time.

5.8.1 Mounting filesystems

First, a few concepts about filesystems. Before a filesystem is accessible to the system, it must be **mounted** on some directory. For example, if you have a filesystem on a floppy, you must mount it under some directory, say `/mnt`, in order to access the files on it (see Section 5.6.2). After mounting the filesystem on that directory, all of the files in the filesystem appear in that directory. In the case of the floppy, the files on the floppy will appear in the directory `/mnt`. After unmounting the floppy, the directory `/mnt` will be empty.

The same is true of filesystems on the hard drive. The system automatically mounts filesystems on your hard drive for you at bootup time. The so-called "root filesystem" is mounted on the directory `/`. If you have a separate filesystem for `/usr`, for example (see Section A.3), it is mounted on `/usr`. If you only have a root filesystem, all files (including those in `/usr`) exist on that filesystem.

The command `mount` is used to mount a filesystem. The command

```
mount -av
```

is executed from the file `/etc/rc` (which is the system initialization file executed at boot time; see Section 5.9.1). The `mount -av` command obtains information on filesystems and mount points from the file `/etc/fstab`. An example `fstab` file appears below.

```
# device       directory      type     options
/dev/hda2       /             ext2     defaults
/dev/hda3       /usr          ext2     defaults
/dev/hda4       none          swap     sw
/proc           /proc         proc     none
```

The first field is the device—the name of the partition to mount. The second field is the mount

point. The third field is the filesystem type—such as `ext2` (for ext2fs) or `minix` (for Minix filesystems). The last field contains `mount` options—usually, this is set to "`default`".

As you can see, swap partitions are included in `/etc/fstab` as well. They have a mount directory of `none`, and type `swap`. The `swapon -a` command, executed from `/etc/rc` as well, is used to enable swapping on all swap devices listed in `/etc/fstab`.

The `fstab` file contains one special entry—for the `/proc` filesystem. As mentioned in Section 4.10.1, the `/proc` filesystem is used to store information about system processes, available memory, and so on. If `/proc` is not mounted, commands such as `ps` will not work.

◇ The `mount` command may only be used by root. This is to ensure security on the system; you wouldn't want regular users mounting and unmounting filesystems on a whim. There are several software packages available which allow regular users to mount and unmount filesystems (floppies in particular) without compromising system security.

The `mount -av` command actually mounts all filesystems other than the root filesystem (in the table above, `/dev/hda2`). The root filesystem is automatically mounted at boot time by the kernel.

Instead of using `mount -av`, you can mount a filesystem by hand. The command

```
# mount -t ext2 /dev/hda3 /usr
```

is equivalent to mounting the filesystem with the entry `/dev/hda3` in the `fstab` example file above.

In general, you should never have to mount or unmount filesystems by hand. The `mount -av` command in `/etc/rc` takes care of mounting the filesystems at boot time. Filesystems are automatically unmounted by the `shutdown` or `halt` commands before bringing the system down.

5.8.2 Checking filesystems

It is usually a good idea to check your filesystems for damage or corrupt files every now and then. Some systems automatically check their filesystems at boot time (with the appropriate commands in `/etc/rc`).

The command used to check a filesystem depends on the type of the filesystem in question. For ext2fs filesystems (the most commonly used type), this command is `e2fsck`. For example, the command

```
# e2fsck -av /dev/hda2
```

will check the ext2fs filesystem on `/dev/hda2` and automatically correct any errors.

It is usually a good idea to unmount a filesystem before checking it. For example, the command

```
# umount /dev/hda2
```

will unmount the filesystem on `/dev/hda2`, after which you can check it. The one exception is that you cannot unmount the root filesystem. In order to check the root filesystem when it's unmounted,

you should use a maintenance boot/root diskette (see Section 5.10.1). You also cannot unmount a filesystem if any of the files in it are "busy"—that is, being used by a running process. For example, you cannot unmount a filesystem if any user's current working directory is on that filesystem. You will receive a "`Device busy`" error if you attempt to unmount a filesystem which is in use.

Other filesystem types use different forms of the `e2fsck` command, such as `efsck` and `xfsck`. On some systems, you can simply use the command `fsck`, which will determine the filesystem type and execute the appropriate command.

◇ It is important that you reboot your system immediately after checking a filesystem is any corrections were made to that filesystem. For example, if `e2fsck` reports that it corrected any errors with the filesystem, you should immediately `shutdown -r` in order to reboot the system. This is to allow the system to re-sync its information about the filesystem when `e2fsck` modifies it.

Of course, the `/proc` filesystem never needs to be checked in this manner. `/proc` is a memory filesystem, managed directly by the kernel.

5.9 Miscellaneous Tasks

Believe it or not, there are a number of housekeeping tasks for the system administrator which don't fall into any major category.

5.9.1 System startup files

When the system boots, a number of scripts are executed automatically by the system before any user logs in. Here is a description of what happens.

At bootup time, the kernel spawns the process `/etc/init`. `init` is a program which reads its configuration file, `/etc/inittab`, and spawns other processes based on the contents of this file. One of the important processes started from `inittab` is the `/etc/getty` process started on each virtual console. The `getty` process grabs the VC for use, and starts a `login` process on the VC. This allows you to login on each VC; if `/etc/inittab` does not contain a `getty` process for a certain VC, you will not be able to login on that VC.

Another process executed from `/etc/inittab` is `/etc/rc`, the main system initialization file. This file is a simple shell script which executes any initialization commands needed at boot time, such as mounting the filesystems (see Section 5.8) and initializing swap space.

Your system may execute other initialization scripts as well, such as `/etc/rc.local`. `/etc/rc.local` usually contains initialization commands specific to your own system, such as setting the hostname (see the next section). `rc.local` may be started from `/etc/rc` or from `/etc/inittab` directly.

5.9.2 Setting the hostname

In a networked environment, the hostname is used to uniquely identify a particular machine, while in a standalone environment the hostname just gives the system personality and charm. It's like naming a pet: you can always address to your dog as "The dog," but it's much more interesting to assign the dog a name such as Spot or Woofie.

Setting the system's hostname is a simple matter of using the `hostname` command. If you are on a network, your hostname should be the full hostname of your machine, such as `goober.norelco.com`. If you are not on a network of any kind, you can choose an arbitrary host and domainname, such as `loomer.vpizza.com`, `shoop.nowhere.edu`, or `floof.org`.

When setting the hostname, the hostname must appear in the file `/etc/hosts`, which assigns an IP address to each host. Even if your machine is not on a network, you should include your own hostname in `/etc/hosts`.

For example, if you are not on a TCP/IP network, and your hostname is `floof.org`, simply include the following line in `/etc/hosts`:

```
127.0.0.1       floof.org localhost
```

This assigns your hostname, `floof.org`, to the loopback address 127.0.0.1 (used if you're not on a network). The `localhost` alias is also assigned to this address.

If you are on a TCP/IP network, however, your real IP address and hostname should appear in `/etc/hosts`. For example, if your hostname is `goober.norelco.com`, and your IP address is 128.253.154.32, add the following line to `/etc/hosts`:

```
128.253.154.32       goober.norelco.com
```

If your hostname does not appear in `/etc/hosts`, you will not be able to set it.

To set your hostname, simply use the `hostname` command. For example, the command

```
# hostname -S goober.norelco.com
```

sets the hostname to `goober.norelco.com`. In most cases, the `hostname` command is executed from one of the system startup files, such as `/etc/rc` or `/etc/rc.local`. Edit these two files and change the `hostname` command found there to set your own hostname; upon rebooting the system the hostname will be set to the new value.

5.9.3 Managing file links

Links allow you to give a single file multiple names. Files are actually identified to the system by their **inode number**, which is just the unique filesystem identifier for the file[5]. A directory is actually a listing of inode numbers with their corresponding filenames. Each filename in a directory is a **link** to a particular inode.

[5] The command `ls -i` will display file inode numbers.

5.9.3.1 Hard links

The `ln` command is used to create multiple links for one file. For example, let's say that you have the file `foo` in a directory. Using `ls -i`, we can look at the inode number for this file.

```
# ls -i foo
22192 foo
#
```

Here, the file `foo` has an inode number of 22192 in the filesystem. We can create another link to `foo`, named `bar`:

```
# ln foo bar
```

With `ls -i`, we see that the two files have the same inode.

```
# ls -i foo bar
22192 bar   22192 foo
#
```

Now, accessing either `foo` or `bar` will access the same file. If you make changes to `foo`, those changes will be made to `bar` as well. For all purposes, `foo` and `bar` are the same file.

These links are known as *hard links* because they directly create a link to an inode. Note that you can only hard-link files on the same filesystem; symbolic links (see below) don't have this restriction.

When you delete a file with `rm`, you are actually only deleting one link to a file. If you use the command

```
# rm foo
```

then only the link named `foo` is deleted; `bar` will still exist. A file is only actually deleted on the system when it has no links to it. Usually, files have only one link, so using the `rm` command deletes the file. However, if a file has multiple links to it, using `rm` will only delete a single link; in order to delete the file, you must delete all links to the file.

The command `ls -l` will display the number of links to a file (among other information).

```
# ls -l foo bar
-rw-r--r--   2 root     root          12 Aug  5 16:51 bar \\ \verb-rw-r--r-- 2
root root 12 Aug 5 16:50 foo
#
```

The second column in the listing, "2", specifies the number of links to the file.

As it turns out, a directory is actually just a file containing information about link-to-inode translations. Also, every directory has at least two hard links in it: `.` (a link pointing to itself), and `..` (a link pointing to the parent directory). The root directory (`/`) `..` link just points back to `/`.

5.9.3.2 Symbolic links

Symbolic links are another type of link, which work differently than hard links (as described above). A symbolic link allows you to give a file another name, but it doesn't link the file by inode.

The command `ln -s` will create a symbolic link to a file. For example, if we use the command

```
# ln -s foo bar
```

we will create the symbolic link `bar` pointing to the file `foo`. If we use `ls -i`, we will see that the two files have different inodes, indeed.

```
# ls -i foo bar
22195 bar   22192 foo
#
```

However, using `ls -l`, we see that the file `bar` is a symlink pointing to `foo`.

```
# ls -l foo bar
lrwxrwxrwx   1 root     root           3 Aug  5 16:51 bar -> foo \\
\verb-rw-r--r-- 1 root root 12 Aug 5 16:50 foo
#
```

The permission bits on a symbolic link are not used (they always appear as `rwxrwxrwx`). Instead, the permissions on the symbolic link are determined by the permissions on the target of the symbolic link (in our example, the file `foo`).

Functionally, hard links and symbolic links are similar, but there are some differences. For one thing, you can create a symbolic link to a file which doesn't exist; the same is not true for hard links. Symbolic links are processed by the kernel differently than hard links are, which is just a technical difference but sometimes an important one. Symbolic links are helpful because they identify what file they point to; with hard links, there is no easy way to determine which files are linked to the same inode.

Links are used in many places on the Linux system. Symbolic links are especially important to the shared library images in `/lib`. See Section 5.7.2 for more information.

5.10 What To Do In An Emergency

On some occasions, the system administrator will be faced with the unique problem of recovering from a complete disaster, such as forgetting the root password or trashing filesystems. The best advice is, *don't panic.* Everyone makes stupid mistakes—that's the best way to learn about system administration: the hard way.

Linux is not an unstable version of UNIX. In fact, I have had fewer problems with system hangs and panics than with commercial versions of UNIX on many platforms. Linux also benefits from a strong complement of wizards who can help you get out of a bind.

The first thing you should do when investigating any problem is to attempt to fix it yourself. Poke around, see how things work. Too much of the time, a system administrator will post a desperate plea for help before looking into the problem at all. Most of the time, you'll find that fixing problems yourself is actually very easy. It is the path to guruhood.

There are very few cases where reinstalling the system from scratch is necessary. Many new users accidentally delete some essential system file, and immediately reach for the installation disks. This is not a good idea. Before taking such drastic measures, investigate the problem and ask others to help fix things up. In almost all cases, you can recover your system from a maintenance diskette.

5.10.1 Recovering using a maintenance diskette

One indispensable tool for the system administrator is the so called "boot/root disk"—a floppy which can be booted for a complete Linux system, independent of your hard drive. Boot/root disks are actually very simple—you create a root filesystem on the floppy, place all of the necessary utilities on it, and install LILO and a bootable kernel on the floppy. Another technique is to use one floppy for the kernel and another for the root filesystem. In any case, the result is the same: you are running a Linux system completely from floppy.

The canonical example of a boot/root disk is the SLS `a1` disk[6]. This disk contains a bootable kernel and a root filesystem, all on floppy. Usually, it's used only when installing SLS, but it comes in very handy when doing system maintenance.

The H.J Lu boot/root disk, available from `/pub/Linux/GCC` on `sunsite.unc.edu`, is another example of such a maintenance disk. Or, if you're ambitious, you can create your own boot/root disk. In most cases, however, using a pre-made boot/root disk is much easier and will probably be more complete.

Using a boot/root disk is very simple. Just boot the disk on your system, and login as `root` (usually no password). In order to access the files on your hard drive, you will need to mount your filesystems by hand. For example, the command

```
# mount -t ext2 /dev/hda2 /mnt
```

will mount an ext2fs filesystem on `/dev/hda2` under `/mnt`. Remember that `/` is now on the boot/root disk itself; you need to mount your hard drive filesystems under some directory in order to access the files. Therefore, `/etc/passwd` on your hard drive is now `/mnt/etc/passwd` if you mount your root filesystem on `/mnt`.

5.10.2 Fixing the root password

If you forget your root password, no problem. Just boot the boot/root disk, mount your root filesystem on `/mnt`, and blank out the password field for `root` in `/mnt/etc/password`, as so:

[6] See Section 2.2 for information on downloading the SLS release. For this procedure, you don't need to download the entire SLS release—only the `a1` disk

```
root::0:0:root:/:/bin/sh
```

Now `root` has no password; when you reboot from the hard drive you should be able to login as `root` and reset the password using `passwd`.

Aren't you glad you learned how to use `vi`? On your boot/root disk, other editors such as Emacs probably aren't available, but `vi` should be.

5.10.3 Fixing trashed filesystems

If you somehow trash your filesystems, you can run `e2fsck` (if you use the ext2fs filesystem type, that is) to correct any damaged data on the filesystems from floppy. Other filesystem types use different forms of the `e2fsck` command; see Section 5.8 for details.

When checking your filesystems from floppy, it's best for the filesystems to not be mounted.

5.10.4 Recovering lost files

If you accidentally deleted important files on your system, there's no way to "undelete" them. However, you can copy the relevant files from the floppy to your hard drive. For example, if you deleted `/bin/login` on your system (which allows you to login), simply boot the boot/root floppy, mount the root filesystem on `/mnt`, and use the command

```
# cp -a /bin/login /mnt/bin/login
```

The `-a` option tells `cp` to preserve the permissions on the file(s) being copied.

Of course, if the files you deleted weren't essential system files which have counterparts on the boot/root floppy, you're out of luck. If you made backups, you can always restore from them.

5.10.5 Fixing trashed libraries

If you accidentally trashed your libraries or symbolic links in `/lib`, more than likely commands which depended on those libraries will no longer run (see Section 5.7.2). The easiest solution is to boot your boot/root floppy, mount your root filesystem, and fix the libraries in `/mnt/lib`.

Chapter 6

Advanced Features

This chapter will introduce you to some of the more interesting features of Linux. This assumes that you have at least basic UNIX experience, and understand the information contained in the previous chapters.

The most important aspect of Linux that distinguishes it from other implementations of UNIX is its open design and philosophy. Linux was not developed by a small team of programmers headed by a marketing committee with a single goal in mind. It was developed by an ever-increasing group of hackers, putting what they wanted into a homebrew UNIX system. The types of software and diversity of design in the Linux world is large. Some people dislike this lack of uniformity and conformity—however, some call it one of the strongest qualities of Linux.

6.1 The X Window System

The X Window System is a large and powerful (and somewhat complex) graphics environment for UNIX systems. The original X Windows code was developed at MIT; commercial vendors have since made X the industry standard for UNIX platforms. Virtually every workstation in the world runs some variant of X Windows.

A free port of the MIT X Windows version 11, release 5 (X11R5) for 80386/80486 UNIX systems has been developed by a team of programmers headed by David Wexelblat[1]. The release, known as XFree86, is available for System V/386, 386BSD, and other i386 UNIX implementations, including Linux. It includes all of the required binaries, support files, libraries, and tools.

Configuring and using the X Window System is far beyond the scope of this book. You are encouraged to read *The X Window System User's Guide*, by Valerie Quercia and Tim O'Reilly. See Appendix B for information on this book. In this section, we'll give a general overview of installing and configuring X Windows for Linux, but it is far from complete. The man pages and README files included with the Linux X Windows distribution should be very helpful. Also, the Linux FAQ contains information on setting up X Windows.

[1] David may be reached on the Internet at `dwex@mtgzfs3.att.com`.

6.1.1 Hardware requirements

XFree86 supports a wide range of video controllers and monitors. As of XFree86-1.3, the following SVGA chipsets are supported[2]

Tseng ET3000, ET4000; Western Digital/Paradise PVGA1; Western Digital WD90C00, WD90C10, WD90C11, WD90C30, WD90C31; Genoa GVGA; Trident TVGA8800CS, TVGA8900B, TVGA8900C, TVGA8900CL, TVGA9000; ATI 28800-4, 28800-5, 28800-a; NCR 77C22, 77C22E; Cirrus Logic GLGD5420, CLGD5422, CLGD5424, CLGD5426; and Compaq AVGA.

All of the above are supported in both 256 color and monochrome modes, with the exception of the ATI and Cirrus chipsets, which are only supported in 256 color mode.

The monochrome server also supports generic VGA cards, using 64k of video memory in a single bank, and the Hercules card. On the Compaq AVGA, only 64k of video memory is supported for the monochrome server, and the GVGA has not been tested with more than 64k.

Note that the Diamond SpeedStar 24 (and possibly recent SpeedStar+) boards are *not* supported, even though they use the ET4000. The reason for this is that Diamond has changed the mechanism used to select pixel clock frequencies, and will only release programming information under non-disclosure. Supporting this board would restrict distribution of the sources for XFree86.

A mention must be made of accelerated chipsets. At this point, XFree86 does not support any accelerated chipsets. These include the S3 86Cxxx, the ATI Mach8 and Mach32, the IBM 8514/A, the new Western Digital chipset (on the Diamond SpeedStar 24X), the new Cirrus and Tseng chipsets, and TIGA (TI 340x0). Some of these will be supported in XFree86 2.0. However, there is a separate server, XS3, available for S3 chipset boards (such as the Orchid Fahrenheit).

Local bus cards are supported as well. The suggested setup for XFree86 under Linux is a 486 machine with at least 8 megabytes of RAM, and an ET4000 VESA local bus video card. This is the "generic" setup which is known to work and is quite fast. Of course, you must also have a VESA local bus motherboard in order to use the local bus video card. I have run XFree86 on a 486/50 MHz machine with 8 megs of RAM, and it's as fast or faster than many color workstations running proprietary versions of UNIX and X.

6.1.2 Installing XFree86

The Linux binary distribution of XFree86 can be found on a number of Linux FTP sites. On `sunsite.unc.edu`, it is found in the directory `/pub/Linux/X11/Xfree86-1.3`. (As of the time of this writing, the current version is 1.3; newer versions are released periodically). The binary distribution consists of a number of gzipped tar files, all of which unpack from `/`. Installation is very simple; just unpack these tar files and everything should go into the right place.

The files in the distribution are:

[2] This information is taken from the XFree86-1.3 README file by David Dawes.

Package Name	Description
xf86-1.3-bin.tar.gz	Essential binaries and setup files; required.
xf86-1.3-kit.tar.gz	X Server link kit. Only needed to hack the X Server.
xf86-1.3-man.tar.gz	Man pages.
xf86-1.3-pex.tar.gz	PEX utilities, for the PEX programming environment.
xf86-1.3-prg.tar.gz	X11 include files and libraries. Needed to write X programs.
xf86-1.3-xfn.tar.gz	Extra fonts.

After installing the software, you need to link the shared libraries from `/usr/X386/lib` to `/lib`. One way to do this is with the command:

```
# ln -s /usr/X386/lib/lib*.so.? /lib
```

You may wish to physically copy the files instead.

If you installed XFree86 with from a standard Linux distribution such as SLS or TAMU, then all of the files should already be in the right place.

If you wish to use the SVGA color server, the file `/usr/bin/X11/X` should be linked to `/usr/bin/X11/XF86_SVGA`. If you wish to use the monochrome server instead, relink this file to `XF86_MONO` with the command

```
# ln -sf /usr/bin/X11/XF86_MONO /usr/bin/X11/X
```

The XS3 server for S3-based SVGA cards is on `sunsite.unc.edu` in the directory `/pub/Linux/X11/X-servers/s3`. XS3 has its own instructions for setup and configuration; see the `README` files there for more details.

6.1.3 Configuring XFree86

Setting up XFree86 is not difficult in most cases. Only when you have non-standard hardware will XFree86 configuration give you any problems. However, XFree86 configuration is beyond the scope of this document; here, we'll give you a brief overview of how it works.

The main XFree86 configuration file is `/usr/lib/X11/Xconfig`. This file contains information on your mouse, video card parameters, and so on. The file `Xconfig.sample` is provided with the XFree86 distribution as an example. The `XFree86` man page explains the format of this file in detail.

In general, here's how it works. Your video card is capable of driving a number of "dot clocks" which are simply clock frequencies for your card. In turn, each dot clock has a resolution mode associated with it, such as 640x480 or 1024x768. (You are not restrained to using "standard" resolutions, as we will see). In the `Xconfig` file there exists stanzas for configuring your mouse, keyboard, and so on. There also exists stanzas for each server: `vga256` for the color SVGA server, and `vga2` for the monochrome server.

Under each server stanza are lines to set the virtual resolution, chipset type, and so on for you video card. There is also a `Modes` line which specifies which modes are available on your card. Modes are usually named after their resolution. For example,

```
Modes "640x480" "800x600" "1024x768"
```

Each mode on this line is an index into the `modeDB` stanza at the end of the `Xconfig` file. It is this section of the file which determines the actual video parameters for each mode.

There is also an optional `Clocks` line which you can use to set the available dot clocks for your card. By default, XFree86 will determine the clocks at startup time; however, because clock timing can be thrown off by other programs running on your system, it may be easier to set the clocks in the `Xconfig` file.

The `modeDB` section of the `Xconfig` file is the important part. Each video card and monitor has its own set of timing and sync frequencies for different resolutions. The file `/usr/lib/X11/etc/modeDB.txt` contains a database of some known monitor and video card timing numbers. Many card and monitors use the VESA standard timings included in the sample `Xconfig` file.

There are various other documents in `/usr/lib/X11/etc` which you should read. The file `VideoModes.txt` is a tutorial on hacking your own monitor frequency timings if you simply can't get any of the numbers in `modeDB.txt` to work. There is also a collection of sample `Xconfig` files on `sunsite.unc.edu` in the file `/pub/Linux/X11/Xconfig.tgz`. Also see the `XFree86` man pages for more information.

6.1.4 Starting up X

After configuring the `XConfig` file, you can start the server with the `startx` command. There are a few things to take into consideration first, however.

Make sure that the directory `/usr/bin/X11` is on your path. This directory contains all of the X binaries and the server itself.

Secondly, the X server requires a free VC to enable VC switching[3]. In other words, you must have one of your VC's available, with no `login` process running on it. The easiest way to ensure this is to edit `/etc/inittab` and delete one of the `getty` lines which starts up a login process on each VC. In my `inittab`, for example, I run `getty` on `/dev/tty1` through `/dev/tty7` (that is, VC's 1 through 7), but not on `/dev/tty8`.

When running `startx`, the file `$HOME/.xinitrc` is read. This file is a shell script which contains commands to run after the X server is started. If this file doesn't exist, the file `/usr/lib/X11/xinit/xinitrc` is used as a system-wide default instead. You can use this default file as a sample `.xinitrc` file.

Using X Windows is a large topic, and we won't try to cover it here. Read *The X Window System User's Guide*, or another book on using X, for details. See Appendix B information on this book.

[3] While running X, you can switch back to your text VC's using the keys ctrl-alt-F1 through ctrl-alt-F12. To return to X, simply switch to the VC reserved for XFree86.

6.1.5 Exiting X

Usually, the last client started in `.xinitrc` is the one used to shutdown X cleanly. For example, if the last command in `.xinitrc` is

```
exec twm
```

then killing the `twm` process will result in X shutting down.

However, if you need to immediately kill the X server for some reason, you can use the key combination ctrl-alt-backspace.

6.1.6 Accessing MS-DOS Files

If, for some twisted and bizarre reason, you would have need to access files from MS-DOS, it's quite easily done under Linux.

The usual way to access MS-DOS files is to mount an MS-DOS partition or floppy under Linux, allowing you to access the files directly through the filesystem. For example, if you have an MS-DOS floppy in `/dev/fd0`, the command

```
# mount -t msdos /dev/fd0 /mnt
```

will mount it under `/mnt`. See Section 5.6.2 for more information on mounting floppies.

You can also mount an MS-DOS partition of your hard drive for access under Linux. If you have an MS-DOS partition on `/dev/hda1`, the command

```
# mount -t msdos /dev/hda1 /mnt
```

will mount it. Be sure to `umount` the partition when you're done using it. You can have your MS-DOS partitions automatically mounted at boot time if you include entries for them in `/etc/fstab`; see Section 5.8 for details. For example, the following line in `/etc/fstab` will mount an MS-DOS partition on `/dev/hda1` on the directory `/dos`.

```
/dev/hda1      /dos      msdos      defaults
```

The `Mtools` software may also be used to access MS-DOS files. For example, the commands `mcd`, `mdir`, and `mcopy` all behave as their MS-DOS counterparts. If you installed `Mtools`, there should be man pages available for these commands.

Accessing MS-DOS files is one thing; running MS-DOS programs from Linux is another. There is an MS-DOS Emulator under development for Linux; it is widely available, and even distributed with SLS. It can be retrieved from a number of locations, including the various Linux FTP sites (see Appendix C for details). The MS-DOS Emulator is reportedly powerful enough to run a number of applications, including Wordperfect, from Linux. However, Linux and MS-DOS are vastly different operating systems. The power of any MS-DOS emulator under UNIX is somewhat limited.

In addition, work is underway on a Microsoft Windows emulator to run under X Windows. Watch the newsgroups and FTP sites for more information.

6.2 Networking with TCP/IP

Linux supports a full implementation of the TCP/IP (Transport Control Protocol/Internet Protocol) networking protocols. TCP/IP has become the most successful mechanism for networking computers worldwide. With Linux and an Ethernet card, you can network your machine to a local area network, or (with the proper network connections), to the Internet—the worldwide TCP/IP network.

Hooking up a small LAN of UNIX machines is easy. It simply requires an Ethernet controller in each machine and the appropriate Ethernet cables and other hardware. Or, if your business or university provides access to the Internet, you can easily add your Linux machine to this network.

Linux TCP/IP also supports SLIP—Serial Line Internet Protocol. SLIP allows you to have dialup Internet access using a modem. If your business or university provides SLIP access, you can dial in to the SLIP server and put your machine on the Internet over the phone line. Alternately, if your Linux machine also has Ethernet access to the Internet, you can set up your Linux box as a SLIP server.

For complete information on setting up TCP/IP under Linux, we encourage you to read *the Linux NET-2-FAQ*, available via anonymous FTP from `sunsite.unc.edu`. The NET-2-FAQ is a complete guide to configuring TCP/IP, including Ethernet and SLIP connections, under Linux. The more complete `Linux Network Administration Guide`, from the Linux Documentation Project, should also be available. Also of interest is the book *TCP/IP Network Administration*, by Craig Hunt. It contains complete information on using and configuring TCP/IP on UNIX systems. See Appendix B for more information.

6.2.1 Hardware Requirements

You can use Linux TCP/IP without any networking hardware at all—configuring "loopback" mode allows you to talk to yourself. This is necessary for some applications and games which use the "loopback" network device.

However, if you want to use Linux with an Ethernet TCP/IP network, you need one of the following Ethernet cards: 3com 3c503, 3c503/16; Novell NE1000, NE2000; Western Digital WD8003, WD8013; Hewlett Packard HP27245, HP27247, HP27250. The following clones are reported to work: WD-80x3 clones: LANNET LEC-45; NE2000 clones: Alta Combo, Artisoft LANtastic AE-2, Asante Etherpak 2001/2003, D-Link Ethernet II, LTC E-NET/16 P/N 8300-200-002, Network Solutions HE-203, SVEC 4 Dimension Ethernet, 4-Dimension FD0490 EtherBoard 16, and D-Link DE-600.

Linux also supports SLIP, which allows you to use a modem to access the Internet over the phone line. In this case, you'll need a modem compatible with your SLIP server—most servers require a 14.4bps V.32bis modem, such as the US Robotics Sportster, or the Infotel 144DF Internal.

6.3 Networking with UUCP

UUCP (UNIX-to-UNIX Copy) is an older mechanism used to transfer information between UNIX systems. Using UUCP, UNIX systems dial each other up (using a modem) and transfer mail mes-

sages, news articles, files, and so on. If you don't have TCP/IP or SLIP access, you can use UUCP to communicate with the world. Most of the mail and news software (see Sections 6.4 and 6.5) can be configured to use UUCP to transfer information to other machines. In fact, if there is an Internet site nearby, you can arrange to have Internet mail sent to your Linux machine via UUCP from that site.

The *Linux Network Administrator's Guide* contains complete information on configuring and using UUCP under Linux. Also, the Linux *UUCP-MAIL-NEWS FAQ*, available via anonymous FTP from `sunsite.unc.edu`, should be of help. Another source of information on UUCP is the book *Managing UUCP and USENET*, by Tim O'Reilly and Grace Todino. See Appendix B for more information.

6.4 Electronic Mail

Like most UNIX systems, Linux provides a number of software packages for using electronic mail. E-mail on your system can either be local (that is, you only mail other users on your system), or networked (that is, you mail, using either TCP/IP or UUCP, users on other machines on a network). E-mail software usually consists of two parts: a *mailer* and a *transport*. The mailer is the user-level software which is used to actually compose and read e-mail messages. Popular mailers include `elm` and `mailx`. The transport is the low-level software which actually takes are of delivering the mail, either locally or remotely. The user never sees the transport software; they only interact with the mailer. However, as the system administrator, it is important to understand the concepts behind the transport software and how to configure it.

The most popular transport software for Linux is `Smail`. This software is easy to configure, and is able to send both local and remote TCP/IP e-mail. The more powerful `sendmail` transport is used on most UNIX systems, however, because of its complicated setup mechanism, most Linux systems don't use it.

The Linux *UUCP-MAIL-NEWS FAQ* gives more information on the available mail software for Linux and how to configure it on your system. If you plan to send mail remotely, you'll need to understand either TCP/IP or UUCP, depending on how your machine is networked (see Sections 6.2 and 6.3). The UUCP and TCP/IP documents listed in Appendix B should be of help there.

Most of the Linux mail software can be retrieved via anonymous FTP from `sunsite.unc.edu` in the directory `/pub/Linux/system/Mail`.

6.5 News and USENET

Linux also provides a number of facilities for managing electronic news. You may choose to set up a local news server on your system, which will allow users to post "articles" to various "newsgroups" on the system...a lively form of discussion. However, if you have access to a TCP/IP or UUCP network, then you will be able to participate in USENET—a worldwide network news service.

There are two parts to the news software—the *server* and the *client*. The news server is the

software which controls the newsgroups and handles delivering articles to other machines (if you are on a network). The news client, or *newsreader*, is the software which connects to the server to allow users to read and post news.

There are several forms of news servers available for Linux. They all follow the same basic protocols and design. The two primary versions are "C News" and "INN". There are many types of newsreaders, as well, such as `rn` and `tin`. The choice of newsreader is more or less a matter of taste; all newsreaders should work equally well with different versions of the server software. That is, the newsreader is independent of the server software, and vice versa.

If you only want to run news locally (that is, not as part of USENET), then you will need to run a server on your system, as well as install a newsreader for the users. The news server will store the articles in a directory such as `/usr/spool/news`, and the newsreader will be compiled to look in this directory for news articles.

However, if you wish to run news over the network, there are several options open to you. TCP/IP network-based news uses a protocol known as NNTP (Network News Transmission Protocol). NNTP allows a newsreader to read news over the network, on a remote machine. NNTP also allows news servers to send articles to each other over the network—this is the software upon which USENET is based. Most businesses and universities have one or more NNTP servers set up to handle all of the USENET news for that site. Every other machine at the site runs an NNTP-based newsreader to read and post news over the network via the NNTP server. This means that only the NNTP server actually stores the news articles on disk.

Here are some possible scenarios for news configuration.

- You run news locally. That is, you have no network connection, or no desire to run news over the network. In this case, you need to run C News or INN on your machine, and install a newsreader to read the news locally.
- You have access to a TCP/IP network and an NNTP server. If your organization has an NNTP news server set up, you can read and post news from your Linux machine by simply installing an NNTP-based newsreader. (Most newsreaders available can be configured to run locally or use NNTP). In this case, you do not need to install a news server or store news articles on your system. The newsreader will take care of reading and posting news over the network. Of course, you will need to have TCP/IP configured and have access to the network (see Section 6.2).
- You have access to a TCP/IP network but have no NNTP server. In this case, you can run an NNTP news server on your Linux system. You can install either a local or an NNTP-based newsreader, and the server will store news articles on your system. In addition, you can configure the server to communicate with other NNTP news servers to transfer news articles.
- You want to transfer news using UUCP. If you have UUCP access (see Section 6.3), you can participate in USENET as well. You will need to install a (local) news server and a news reader. In addition, you will need to configure your UUCP software to periodically transfer news articles to another nearby UUCP machine (known as your "news feed"). UUCP does not use NNTP to transfer news; simply, UUCP provides its own mechanism for transferring news articles.

The one downside of most news server and newsreader software is that it must be compiled by hand. Most of the news software does not use configuration files; instead, configuration options are determined at compile time.

Most of the "standard" news software (available via anonymous FTP from `ftp.uu.net` in the directory `/news`) will compile out-of-the box on Linux. Necessary patches can be found on `sunsite.unc.edu` in `/pub/Linux/system/Mail` (which is, incidentally, also where mail software for Linux is found). Other news binaries for Linux may be found in this directory as well.

For more information, refer to the Linux *UUCP-MAIL-NEWS FAQ* from `sunsite.unc.edu` in `/pub/Linux/docs`. Also, the LDP's *Linux Network Administrator's Guide* contains complete information on configuring news software for Linux. The book *Managing UUCP and Usenet*, by Tim O'Reilly and Grace Todino, is an excellent guide to setting up UUCP and news software. Also of interest is the USENET document "How to become a USENET site," available from `ftp.uu.net`, in the directory `/usenet/news.announce.newusers`.

Appendix A

Tips, Tricks, and Common Problems

This appendix contains information on some commonly used tricks and techniques, as well as solutions to common problems.

A.1 Installing SLS From the Hard Drive

Instead of installing SLS from floppy, you can simply keep all of the distribution files on another partition on your hard drive (say, your MS-DOS partition), and install from there.

First, you need to create the SLS `a1` disk as you would for a floppy installation. Secondly, all of the other SLS files need to go in a directory named `install` at the top level of the partition. For example, if you plan to install from an MS-DOS partition on drive `C:`, the directory names will be `C:\install\a2`, `C:\install\a3`, and so on. Each subdirectory of `install` will contain the files for the corresponding diskette.

Make sure that `install` is a top-level directory on the partition. In other words, the directory `C:\SLS\install` will not work. It must be `C:\install`.

All of the other installation steps (creating partitions and filesystems, and so on) are identical to those found in Chapter 3. When using the `doinstall` command, simply choose the menu option to install from the hard drive. You will be prompted for the partition name (such as `/dev/hda2`) and the type of partition (such as `msdos`). Everything else should be self-explanatory.

A.2 Installing the SLS CD-ROM

If someone will lend me a CD-ROM drive, I can write this section.

A.3 Using Multiple Filesystems

If you have experience with UNIX system administration, it should be easy to decide how to use multiple filesystems with Linux. For example, you can use separate filesystems for root and `/usr` space.

There is no inherent benefit to using multiple filesystems for Linux. However, if you're conscious about keeping everything mounted at once, then you may wish to do so. For example, if you accidentally trash your root filesystem, then other filesystems, such as `/usr`, won't be affected. However, using multiple filesystems means that you have to plan your space carefully—there is no way to nondestructively resize filesystems (yet).

You may wish to use multiple swap partitions as well—because swap partitions are limited to 16 megabytes each. You may use up to 8 swap partitions on your system.

Using multiple filesystems is easy. First of all, you must have a separate partition for each filesystem. If you are creating a number of filesystems, you may need to use logical partitions in order to overcome the 4 primary partition limit. While using `fdisk`, simply use the "`n`" command to create a new partition for each Linux filesystem you want to create.

Before installing the software, simply use the appropriate `mke2fs` command for each filesystem that you wish to create.

When installing the software, there may be extra steps involved in getting the system to recognize all of your filesystems for installation. For some releases, you may need to mount all of the filesystems in the appropriate place.

To install the SLS release, all you need to do is use extra arguments on the `doinstall` command, as so:

`doinstall` ⟨*rootfs*⟩ ⟨*fs1*⟩ ⟨*mount-pt1*⟩ ⟨*fs2*⟩ ⟨*mount-pt2*⟩ ... ⟨*fsN*⟩ ⟨*mount-ptN*⟩

For example, if you have your root filesystem on `/dev/hda2`, your `/usr` filesystem on `/dev/hda3`, and your `/tmp` filesystem on `/dev/hda4`, use the command

```
# doinstall /dev/hda2 /dev/hda3 /usr /dev/hda4 /tmp
```

See your documentation for details on using multiple filesystems with the release of Linux that you're installing.

A.4 Using `dd` Instead of `rawrite.exe`

If you would rather create your Linux install disks from a UNIX system, rather than using `rawrite.exe` on an MS-DOS system, you can use the following procedure. Note that this procedure does not specify how to create MS-DOS formatted disks (such as the SLS `a2` through `a4` disks). This procedure only replaces the use of `rawrite.exe` (for the SLS `a1` disk, for example).

The `dd` command may be used on a UNIX system to replace the use of `rawrite`. The format of the command is as follows:

```
dd if=⟨input-file⟩ of=⟨output-file⟩ bs=16k conv=sync
```

The ⟨*input-file*⟩ is the name of the file to write to the floppy, such as "`a1.3`".

⟨*output-file*⟩ is the name of the floppy device to write the file to. On many systems, this is `/dev/rfd`*n*, where *n* is a number such as `0` or `1` depending on the unit number of the floppy drive. However, this varies greatly from system to system. You need to check with the system administrator of the system in question to determine the device name of the floppy drive. For example, on some systems the following will work:

```
# dd if=a1.3 of=/dev/rfd0 bs=16k conv=sync
```

Under Linux, the floppy drive device name is either `/dev/fd0` (for the first floppy drive), or `/dev/fd1` (for the second). You can use the command

```
# dd if=a1.3 of=/dev/fd0 bs=16k conv=sync
```

to "rawrite" a floppy from another Linux system. Or, you simply use the `cp` command as so:

```
# cp a1.3 /dev/fd0
```

`cp` may work on other UNIX systems as well.

A.5 Using a swap file

Instead of reserving an individual partition for swap space, you can use a file. However, to do so you'll need install the Linux software and get everything going *before* you create the swap file.

If you have a Linux system installed, you can use the following commands to create a swap file. Below, we're going to create a swap file of size 8208 blocks (about 8 megs).

```
# dd if=/dev/zero of=/swap bs=1024 count=8208
```

This command creates the swap file itself. Replace the "`count=`" with the size of the swap file in blocks.

```
# swapon /swap
```

Now we are swapping on the file `/swap` which we have created.

The one major drawback to using a swapfile in this manner is that all access to the swap file is done through the filesystem. This means that the blocks which make up the swap file may not be

contiguous. Therefore, performance may not be as great as using a swap partition, for which blocks are always contiguous and I/O requests are done directly to the device.

Another drawback in using a swapfile is the chance to corrupt your filesystem data—when using large swap files, there is the chance that you can corrupt your filesystem if something goes wrong. Keeping your filesystems and swap partitions separate will prevent this from happening.

Using a swap file can be very useful if you have a temporary need for more swap space. For example, if you're compiling a large program and would like to speed things up somewhat, you can temporarily create a swap file and use it in addition to your regular swap space.

To get rid of a swap file, first use **swapoff**, as in

```
# swapoff /swap
```

And you can safely delete the file.

```
# rm /swap
```

Remember that each swap file (or partition) may be as large as 16 megabytes, but you may use up to 8 swap files or partitions on your system.

Appendix B

Bibliography and Sources of Information

This appendix provides a listing of other valuable sources of information about Linux, including recommended UNIX books which would be of use to the new Linux user.

B.1 The Linux Frequently Asked Questions List

The Linux FAQ is a listing of frequently asked questions and answers about Linux. It is a large document which covers almost every problem relevant to the Linux community. However, the current Linux FAQ is a large, bloated document which is in dire need of reorganization. Matt Welsh and Ian Jackson are currently working on a complete rewrite of the Linux FAQ. Hopefully it will be complete by the time you read this manual!

The Linux FAQ can be retrieved from a number of FTP sites worldwide. The canonical location is `sunsite.unc.edu`, in the file `/pub/Linux/docs/FAQ`. The directory `/pub/Linux/docs/faqs` contains other versions of the FAQ, including texinfo and `.dvi` format. You can find a number of other Linux documents in this directory as well.

The Linux FAQ is also posted monthly to the USENET newsgroups `comp.os.linux.help`, `comp.os.linux.announce`, and `news.answers`. If you don't have USENET or FTP access, you can retrieve the Linux FAQ via the mail server at `rtfm.mit.edu`. Send mail to the address

```
mail-server@rtfm.mit.edu
```

with the word "`help`" in the body.

It is also distributed as part of several Linux releases, such as SLS and the Yggdrasil CD-ROM.

If you simply can't find a copy of the Linux FAQ any other way, feel free to mail Matt Welsh at `mdw@sunsite.unc.edu`; I can send it to you directly.

B.2 The Linux Documentation Project Manuals

The Linux Documentation Project is in the process of providing a number of manuals and books for Linux (including this one!). The current location for publicly-available LDP manuals is `/pub/linux/ALPHA/LDP` on `tsx-11.mit.edu`. See the file `INDEX` for a listing of what's available.

The LDP manuals currently under development are: *The Linux System Administrator's Guide*, *Linux Network Administrator's Guide*, *Linux Kernel Hacker's Guide*, *Linux User's Guide*, and, of course, *Linux Installation and Getting Started*. All of these are in various states of completion.

Keep in mind the manuals in this directory are under development—that is, there may be parts and sections still to be written, and everything may not be completely accurate. Once the manuals are "complete", they will be moved to another location.

B.3 Other Online Documents

There are a number of other online documents of interest to new Linux users.

B.3.1 The Linux INFO-SHEET

The INFO-SHEET is a short document giving some of the technical details of Linux, including hardware requirements and other information.
You can find it on `sunsite.unc.edu:/pub/Linux/docs/INFO-SHEET`. It is posted to the various Linux USENET newsgroups as well.

B.3.2 The Linux META-FAQ

The Linux META-FAQ is a collection of Linux sources of information (much like this appendix). It serves as a general introduction to getting Linux information; most of the material therein is covered in this manual. This file can be found on `sunsite.unc.edu:/pub/Linux/docs/META-FAQ`.

B.3.3 The Linux Hardware Compatibility List

The Hardware Compatibility List lists some of the hardware which is known to work with Linux. However, much of the hardware is not on this list; for example, all IDE drives should work with Linux, so all makes and models are not listed there. The Hardware Compatibility List is in the file `sunsite.unc.edu:/pub/Linux/docs/HARDWARE`.

B.3.4 The Linux Software Map

The Linux Software Map is a listing of some of the available software for Linux, where to get it, and who maintains it. However, the LSM is far from complete; most of the people who distribute

software for Linux haven't added entries to it (also, the LSM project is relatively new). Therefore, keep in mind that there is much more software available for Linux than is actually listed in the LSM.

The current version of the LSM can be found on `sunsite.unc.edu` in the file `/pub/Linux/docs/LSM.z`. This is a gzipped file; you must use the `gzip` utility to uncompress it.

B.3.5 The Linux HOWTO Documents

The Linux HOWTOs are a collection of "how to" documents, each describing a certain aspect of the system. For example, the Installation HOWTO explains how to obtain and install Linux (an extreme condensation of this book), the NET-2-HOWTO explains how to configure TCP/IP under Linux, the Mail HOWTO describes configuration of electronic mail, and so on.

Linux HOWTOs can be found on many Linux FTP sites, including `sunsite.unc.edu` in the directory `/pub/Linux/docs/HOWTO`. The file `HOWTO-INDEX` includes a list of currently available HOWTOs.

B.3.6 Others

On `sunsite.unc.edu:/pub/Linux/docs` are a number of other documents related to Linux. Most of these aren't of concern to the new user; all of the introductory and installation material for Linux is covered in this manual.

B.4 Linux USENET Newsgroups

There are two USENET newsgroups are devoted to Linux. These are `comp.os.linux.help`, for general questions about using and running Linux; `comp.os.linux.admin`, for discussions about administration of Linux systems; `comp.os.linux.development`, for kernel and systems-level Linux programming, and `comp.os.linux.misc`, for miscellaneous discussions about Linux. Also, there is the newsgroups `comp.os.linux.announce`, which is a moderated newsgroup for Linux announcements and bug patches. Submissions to `comp.os.linux.announce` must be approved by the moderator, Matt Welsh, before they are posted. When posting to `c.o.l.a`, your news software will usually automatically mail the posting to the moderator for approval. However, if your news software is not configured correctly, you may have to mail the posting directly to the address `linux-announce@tc.cornell.edu`.

Postings in `comp.os.linux.announce` are archived on `sunsite.unc.edu` in the directory `/pub/Linux/docs/linux-announce.archive`. Traffic to `c.o.l.a` is also mirrored to the "`ANNOUNCE`" channel of the `linux-activists` mailing list; see the next section for details.

It is strongly suggested that you read the Linux FAQ and read the newsgroup befores posting any questions—more than likely, your question has already been covered in the newsgroup.

B.5 Linux Mailing Lists

There are a number of mailing lists available for the Linux community as well. The most prominent of these is the `linux-activists` mailing list. This list is generally for developers and programmers for Linux—there are very few discussions relating to using or installing Linux. All such user questions about Linux should be directed to the newsgroup `comp.os.linux.help` instead.

The `linux-activists` mailing list is a multi-channel mailing list; that is, you join a particular "channel" and receive mail (usually in digest format) for that channel only. You can join multiple channels if you wish. For more information on joining this mailing list, send mail to

```
linux-announce-request@niksula.hut.fi
```

with the word "`help`" in the subject.

B.6 Bibliography

At present, the only books devoted to Linux are those distributed by the Linux Documentation Project. However, many of the books available for other versions of UNIX are relevant to Linux as well.

Title: *Learning the UNIX Operating System*
Author: Grace Todino & John Strang
Publisher: O'Reilly and Associates, 1987
ISBN: 0-937175-16-1, $9.00

A good introductory book on learning the UNIX operating system. Most of the information should be applicable to Linux as well. I suggest reading this book if you're new to UNIX and really want to get started with using your new system.

Title: *Learning the* `vi` *Editor*
Author: Linda Lamb
Publisher: O'Reilly and Associates, 1990
ISBN: 0-937175-67-6, $21.95

In Section 4.11 we introduced some of the basic aspects of the `vi` editor. This book will teach you the rest. It's important to know `vi` if you're serious about UNIX; you're not always going to have access to a "real" editor such as Emacs.

Title: *Essential System Administration*

Author: Æleen Frisch
Publisher: O'Reilly and Associates, 1991
ISBN: 0-937175-80-3, $29.95

From the O'Reilly and Associates Catalog, "Like any other multi-user system, UNIX requires some care and feeding. *Essential System Administration* tells you how. This book strips away the myth and confusion surrounding this important topic and provides a compact, manageable introduction to the tasks faced by anyone responsible for a UNIX system." I couldn't have said it better myself.

Title: *TCP/IP Network Administration*
Author: Craig Hunt
Publisher: O'Reilly and Associates, 1990
ISBN: 0-937175-82-X, $24.95

A complete guide to setting up and running a TCP/IP network. While this book is not Linux-specific, roughly 90% of it is applicable to Linux. Coupled with the Linux NET-2-FAQ and *Linux Network Administrator's Guide*, this is a great book discussing the concepts and technical details of managing TCP/IP.

Title: *Managing UUCP and Usenet*
Author: Tim O'Reilly and Grace Todino
Publisher: O'Reilly and Associates, 1991
ISBN: 0-937175-93-5, $24.95

This book covers how to install and configure UUCP networking software, including configuration for USENET news. If you're at all interested in using UUCP or accessing USENET news on your system, this book is a must-read. See Sections 6.3 and 6.5 for more information on UUCP and USENET for Linux.

Title: *Practical UNIX Security*
Author: Simson Garfinkel & Gene Spafford
Publisher: O'Reilly and Associates, 1991
ISBN: 0-937175-72-2, $29.95

This is an excellent book on UNIX system security. It taught me quite a few things that I didn't know, even with several years of UNIX system administration experience. As most UNIX books, this book is geared for large systems, but almost all of the content is relevant to Linux.

Title: *X Window System User's Guide*
Author: Valerie Quercia & Tim O'Reilly
Publisher: O'Reilly and Associates, 1993
ISBN: 1-56592-014-7, $34.95

A complete tutorial and reference guide to using the X Window System. If you installed X windows on your Linux system, and want to know how to get the most out of it, you should read this book. Unlike some windowing systems, a lot of the power provided by X is not obvious as first sight. After using X Windows for several years I learned some things by reading through this book.

Title: *Using C on the UNIX System*
Author: Dave Curry
Publisher: O'Reilly and Associates, 1989
ISBN: 0-937175-23-4, $24.95

This book will introduce you to all of the aspects of programming on the UNIX system call level. It covers everything from files to terminal I/O to TCP/IP sockets. If you want to develop software for Linux, this book is a must read.

Appendix C

FTP Tutorial and Site List

FTP ("File Transfer Protocol") is the set of programs that are used for transferring files between systems on the Internet. Most UNIX, VMS, and MS-DOS systems on the Internet have a program called `ftp` which you use to transfer these files, and if you have Internet access, the best way to download the Linux software is by using `ftp`. This appendix covers basic `ftp` usage—of course, there are many more functions and uses of `ftp` than are given here. Chapter 2 tells you how to use FTP to download the Linux software.

At the end of this appendix there is a listing of FTP sites where Linux software can be found.

If you're using an MS-DOS, UNIX, or VMS system to download files from the Internet, then `ftp` is a command-driven program. However, there are other implementations of `ftp` out there, such as the Macintosh version (called `Fetch`) which has a nice menu-driven interface, which is quite self-explanatory. Even if you're not using the command-driven version of `ftp`, the information given here should help.

`ftp` can be used to both upload (send) or download (receive) files from other Internet sites. In most situations, you're going to be downloading software. On the Internet there are a large number of publicly-available **FTP archive sites**, machines which allow anyone to `ftp` to them and download free software. One such archive site is `sunsite.unc.edu`, which has a lot of Sun Microsystems software, and acts as one of the main Linux sites. In addition, FTP archive sites **mirror** software to each other—that is, software uploaded to one site will be automatically copied over to a number of other sites. So don't be surprised if you see the exact same files on many different archive sites.

C.1 Starting `ftp`

Before you start `ftp` you need a site to connect to. Note that in the example "screens" printed below I'm only showing the most important information, and what you see may differ. Also, commands in *italics* represent commands that you type; everything else is screen output.

To start `ftp` and connect to a site, give simply use the command

`ftp` ⟨*hostname*⟩

where ⟨*hostname*⟩ is the name of the site you are connecting to. For example, to connect to the mythical site `shoop.vpizza.com` we can use the command

```
ftp shoop.vpizza.com
```

C.2 Logging In

When `ftp` starts up we should see something like

```
Connected to shoop.vpizza.com.
220 Shoop.vpizza.com FTPD ready at 15 Dec 1992 08:20:42 EDT
Name (shoop.vpizza.com:mdw):
```

Here, `ftp` is asking us to give the username that we want to login as on `shoop.vpizza.com`. The default here is `mdw`, which is my username on the system I'm using FTP from. Since I don't have an account on `shoop.vpizza.com` I can't login as myself. Instead, to access publicly-available software on an FTP site you login as `anonymous`, and give your Internet e-mail address (if you have one) as the password. So, we would type

```
Name (shoop.vpizza.com:mdw):  anonymous
331-Guest login ok, send e-mail address as password.
Password:  mdw@sunsite.unc.edu
230- Welcome to shoop.vpizza.com.
230- Virtual Pizza Delivery[tm]:  Download pizza in 30 cycles or less
230- or you get it FREE!
ftp>
```

Of course, in you should give your e-mail address, instead of mine, and it won't echo to the screen as you're typing it (since it's technically a "password"). `ftp` should allow us to login and we'll be ready to download software.

C.3 Poking Around

Okay, we're in. `ftp>` is our prompt, and the `ftp` program is waiting for commands. There are a few basic commands you need to know about. First, the commands

`ls` ⟨*file*⟩

and

`dir` ⟨*file*⟩

both give file listings (where ⟨*file*⟩ is an optional argument specifying a particular filename to list). The difference is that `ls` usually gives a short listing and `dir` gives a longer listing (that is, with more information on the sizes of the files, dates of modification, and so on).

The command

`cd` ⟨*directory*⟩

will move to the given directory (just like the `cd` command on UNIX or MS-DOS systems). You can use the command

```
cd ..
```

to change to the parent directory[1]. Note the space between the "`cd`" and the "`..`".

The command

`help` ⟨*command*⟩

will give help on the given `ftp` ⟨*command*⟩ (such as `ls` or `cd`). If no command is specified, `ftp` will list all of the available commands.

If we type `dir` at this point we'll see an initial directory listing of where we are.

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 1337


dr-xr-xr-x  2 root     wheel         512 Aug 13 13:55 bin
drwxr-xr-x  2 root     wheel         512 Aug 13 13:58 dev
drwxr-xr-x  2 root     wheel         512 Jan 25 17:35 etc
drwxr-xr-x 19 root     wheel        1024 Jan 27 21:39 pub
drwxrwx-wx  4 root     ftp-admi     1024 Feb  6 22:10 uploads
drwxr-xr-x  3 root     wheel         512 Mar 11  1992 usr

226 Transfer complete.
921 bytes received in 0.24 seconds (3.7 Kbytes/s)
ftp>
```

Each of these entries is a directory, not an individual file which we can download (specified by the `d` in the first column of the listing). On most FTP archive sites, the publicly available software is under the directory `/pub`, so let's go there.

```
ftp> dir
200 PORT command successful.
```

[1] The directory above the current one.

```
150 ASCII data connection for /bin/ls (128.84.181.1,4525) (0 bytes).
total 846

-rw-r--r--   1 root     staff        1433 Jul 12  1988 README
-r--r--r--   1 3807     staff       15586 May 13  1991 US-DOMAIN.TXT.2
-rw-r--r--   1 539      staff       52664 Feb 20  1991 altenergy.avail
-r--r--r--   1 65534    65534       56456 Dec 17  1990 ataxx.tar.Z
-rw-r--r--   1 root     other     2013041 Jul  3  1991 gesyps.tar.Z
-rw-r--r--   1 432      staff       41831 Jan 30  1989 gnexe.arc
-rw-rw-rw-   1 615      staff       50315 Apr 16  1992 linpack.tar.Z
-r--r--r--   1 root     wheel       12168 Dec 25  1990 localtime.o
-rw-r--r--   1 root     staff        7035 Aug 27  1986 manualslist.tblms
drwxr-xr-x   2 2195     staff         512 Mar 10 00:48 mdw
-rw-r--r--   1 root     staff        5593 Jul 19  1988 t.out.h

226 ASCII Transfer complete.
2443 bytes received in 0.35 seconds (6.8 Kbytes/s)
ftp>
```

Here we can see a number of (interesting?) files, one of which is called `README`, which we should download (most FTP sites have a `README` file in the `/pub` directory).

C.4 Downloading files

Before downloading files, there are a few things that you need to take care of.

- **Turn on hash mark printing.** *Hash marks* are printed to the screen as files are being transferred; they let you know how far along the transfer is, and that your connection hasn't hung up (so you don't sit for 20 minutes, thinking that you're still downloading a file). In general, a hash mark appears as a pound sign (`#`), and one is printed for every 1024 or 8192 bytes transferred, depending on your system.

  To turn on hash mark printing, give the command `hash`.

  ```
  ftp> hash
  Hash mark printing on (8192 bytes/hash mark).
  ftp>
  ```

- **Determine the type of file which you are downloading.** As far as FTP is concerned, files come in two flavors: *binary* and *text*. Most of the files which you'll be downloading are binary files: that is, programs, compressed files, archive files, and so on. However, many files (such as `README`s and so on) are text files.

  Why does the file type matter? Only because on some systems (such as MS-DOS systems), certain characters in a text file, such as carriage returns, need to be converted so that the file

will be readable. While transferring in binary mode, no conversion is done—the file is simply transferred byte after byte.

The commands `bin` and `ascii` set the transfer mode to binary and text, respectively. *When in doubt, always use binary mode to transfer files.* If you try to transfer a binary file in text mode, you'll corrupt the file and it will be unusable. (This is one of the most common mistakes made when using FTP.) However, you can use text mode for plain text files (whose filenames often end in `.txt`).

For our example, we're downloading the file `README`, which is most likely a text file, so we use the command

```
ftp> ascii
200 Type set to A.
ftp>
```

- **Set your local directory.** Your *local directory* is the directory on your system where you want the downloaded files to end up. Whereas the `cd` command changes the remote directory (on the remote machine which you're FTPing to), the `lcd` command changes the local directory.

  For example, to set the local directory to `/home/db/mdw/tmp`, use the command

```
ftp> lcd /home/db/mdw/tmp
Local directory now /home/db/mdw/tmp
ftp>
```

Now you're ready to actually download the file. The command

`get` ⟨*remote-name*⟩ ⟨*local-name*⟩

is used for this, where ⟨*remote-name*⟩ is the name of the file on the remote machine, and ⟨*local-name*⟩ is the name that you wish to give the file on your local machine. The ⟨*local-name*⟩ argument is optional; by default, the local filename is the same as the remote one. However, if for example you're downloading the file `foo.txt`, and you already have a `foo.txt` in your local directory, you'll want to give a different ⟨*local-filename*⟩ so that the first one isn't overwritten.

For our example, to download the file `README`, we simply use

```
ftp> get README
200 PORT command successful.
150 ASCII data connection for README (128.84.181.1,4527) (1433 bytes).
#
226 ASCII Transfer complete.
local:  README remote:  README
1493 bytes received in 0.03 seconds (49 Kbytes/s)
ftp>
```

C.5 Quitting FTP

To end your FTP session, simply use the command

```
quit
```

The command

```
close
```

can be used to close the connection with the current remote FTP site; the `open` command can then be used to start a session with another site (without quitting the FTP program altogether).

```
ftp> close
221 Goodbye.
ftp> quit
```

C.6 Linux FTP Site List

Table C.1 is a listing of the most well-known FTP archive sites which carry the Linux software. Keep in mind that many other sites mirror these, and more than likely you'll run into Linux on a number of sites not on this list.

Site name	IP Address	Directory
tsx-11.mit.edu	18.172.1.2	/pub/linux
sunsite.unc.edu	152.2.22.81	/pub/Linux
nic.funet.fi	128.214.6.100	/pub/OS/Linux
ftp.mcc.ac.uk	130.88.200.7	/pub/linux
fgb1.fgb.mw.tu-muenchen.de	129.187.200.1	/pub/linux
ftp.informatik.tu-muenchen.de	131.159.0.110	/pub/Linux
ftp.dfv.rwth-aachen.de	137.226.4.105	/pub/linux
ftp.informatik.rwth-aachen.de	137.226.112.172	/pub/Linux
ftp.ibp.fr	132.227.60.2	/pub/linux
kirk.bu.oz.au	131.244.1.1	/pub/OS/Linux
ftp.uu.net	137.39.1.9	/systems/unix/linux
wuarchive.wustl.edu	128.252.135.4	/systems/linux
ftp.win.tue.nl	131.155.70.100	/pub/linux
ftp.stack.urc.tue.nl	131.155.2.71	/pub/linux
ftp.ibr.cs.tu-bs.de	134.169.34.15	/pub/os/linux
ftp.denet.dk	129.142.6.74	/pub/OS/linux

Table C.1: Linux FTP Sites

`tsx-11.mit.edu`, `sunsite.unc.edu`, and `nic.funet.fi` are the "home sites" for the Linux software, where most of the new software is uploaded. Most of the other sites on the list mirror some

combination of these three. To reduce network traffic, choose a site which is geographically closest to you.

Appendix D

Linux BBS List

Printed here is a list of bulliten board systems (BBS) which carry Linux software. See Section 2.3 for information on downloading Linux from BBSs.

Zane Healy (`healyzh@holonet.net`) maintains this list. If you know of or run a BBS which provides Linux software which isn't on this list, you should get in touch with him.

The Linux community is no longer an Internet-only society. In fact, it is now estimated that the majority of Linux users don't have Internet access. Therefore, it is especially important that BBSs continue to provide and support Linux to users worldwide.

United States

Citrus Grove Public Access, 916-381-5822. ZyXEL 16.8/14.4 Sacramento, CA. Internet: `citrus.sac.ca.us`
Higher Powered BBS, 408-737-7040. ? CA. RIME ->HIGHER
hip-hop, 408-773-0768. 19.2k Sunnyvale, CA. USENET access
hip-hop, 408-773-0768. 38.4k Sunnyvale, CA.
Unix Online, 707-765-4631. 9600 Petaluma, CA. USENET access
The Outer Rim, 805-252-6342. Santa Clarita, CA.
Programmer's Exchange, 818-444-3507. El Monte, CA. Fidonet
Programmer's Exchange, 818-579-9711. El Monte, CA.
Micro Oasis, 510-895-5985. 14.4k San Leandro, CA.
Test Engineering, 916-928-0504. Sacramento, CA.
Slut Club, 813-975-2603. USR/DS 16.8k HST/14.4K Tampa, FL. Fidonet 1:377/42
Lost City Atlantis, 904-727-9334. 14.4k Jacksonville, FL. FidoNet
Aquired Knowledge, 305-720-3669. 14.4k v.32bis Ft. Lauderdale, FL. Internet, UUCP
The Computer Mechanic, 813-544-9345. 14.4k v.32bis St. Petersburg, FL. Fidonet, Sailnet, MXBBSnet
AVSync, 404-320-6202. Atlanta, GA.
Information Overload, 404-471-1549. 19.2k ZyXEL Atlanta, GA. Fidonet 1:133/308

Atlanta Radio Club, 404-850-0546. 9600 Atlanta, GA.
Rebel BBS, 208-887-3937. 9600 Boise, ID.
Rocky Mountain HUB, 208-232-3405. 38.4k Pocatello, ID. Fionet, SLNet, CinemaNet
EchoMania, 618-233-1659. 14.4k HST Belleville, IL. Fidonet 1:2250/1, f'req LINUX
UNIX USER, 708-879-8633. 14.4k Batavia, IL. USENET, Internet mail
PBS BBS, 309-663-7675. 2400 Bloomington, IL.
Third World, 217-356-9512. 9600 v.32 IL.
Digital Underground, 812-941-9427. 14.4k v.32bis IN. USENET
The OA Southern Star, 504-885-5928. New Orleans, LA. Fidonet 1:396/1
Channel One, 617-354-8873. Boston, MA. RIME ->CHANNEL
VWIS Linux Support BBS, 508-793-1570. 9600 Worcester, MA.
WayStar BBS, 508-481-7147. 14.4k V.32bis USR/HST Marlborough, MA. Fidonet 1:333/14
WayStar BBS, 508-481-7293. 14.4k V.32bis USR/HST Marlborough, MA. Fidonet 1:333/15
WayStar BBS, 508-480-8371. 9600 V.32bis or 14.4k USR/HST Marlborough, MA. Fidonet 1:333/16
Programmer's Center, 301-596-1180. 9600 Columbia, MD. RIME
Brodmann's Place, 301-843-5732. 14.4k Waldorf, MD. RIME ->BRODMANN, Fidonet
Main Frame, 301-654-2554. 9600 Gaithersburg, MD. RIME ->MAINFRAME
1 Zero Cybernet BBS, 301-589-4064. MD.
WaterDeep BBS, 410-614-2190. 9600 v.32 Baltimore, MD.
Harbor Heights BBS, 207-663-0391. 14.4k Boothbay Harbor, ME.
Part-Time BBS, 612-544-5552. 14.4k v.32bis Plymouth, MN.
The Sole Survivor, 314-846-2702. 14.4k v.32bis St. Louis, MO. WWIVnet, WWIVlink, etc
MAC's Place, 919-891-1111. 16.8k, DS modem Dunn, NC. RIME ->MAC
Digital Designs, 919-423-4216. 14.4k, 2400 Hope Mills, NC.
Flite Line, 402-421-2434. Lincoln, NE. RIME ->FLITE, DS modem
Legend, 402-438-2433. Lincoln, NE. DS modem
MegaByte Mansion, 402-551-8681. 14.4 V,32bis Omaha, NE.
Mycroft QNX, 201-858-3429. 14.4k NJ.
Steve Leon's, 201-886-8041. 14.4k Cliffside Park, NJ.
Dwight-Englewood BBS, 201-569-3543. 9600 v.42 Englewood, NJ. USENET
The Mothership Cnection, 908-940-1012. 38.4k Franklin Park, NJ.
The Laboratory, 212-927-4980. 16.8k HST, 14.4k v.32bis NY. FidoNet 1:278/707
Valhalla, 516-321-6819. 14.4k HST v.32 Babylon, NY. Fidonet (1:107/255), UseNet (`die.linet.org`)
Intermittent Connection, 503-344-9838. 14.4k HST v.32bis Eugene, OR. 1:152/35
Horizon Systems, 216-899-1086. USR v.32 Westlake, OH.
Horizon Systems, 216-899-1293. 2400 Westlake, OH.
Centre Programmers Unit, 814-353-0566. 14.4k V.32bis/HST Bellefonte, PA.
Allentown Technical, 215-432-5699. 9600 v.32/v.42bis Allentown, PA. WWIVNet 2578
Tactical-Operations, 814-861-7637. 14.4k V32bis/V42bis State College, PA. Fidonet 1:129/226, `tac_ops.UUCP`
North Shore BBS, 713-251-9757. Houston, TX.
The Annex, 512-575-1188. 9600 HST TX. Fidonet 1:3802/217

The Annex, 512-575-0667. 2400 TX. Fidonet 1:3802/216
Walt Fairs, 713-947-9866. Houston, TX. FidoNet 1:106/18
CyberVille, 817-249-6261. 9600 TX. FidoNet 1:130/78
splat-ooh, 512-578-2720. 14.4k Victoria, TX.
splat-ooh, 512-578-5436. 14.4k Victoria, TX.
alaree, 512-575-5554. 14.4k Victoria, TX.
Ronin BBS, 214-938-2840. 14.4 HST/DS Waxahachie (Dallas), TX. RIME, Intelec, Smartnet, etc.
VTBBS, 703-231-7498. Blacksburg, VA.
MBT, 703-953-0640. Blacksburg, VA.
NOVA, 703-323-3321. 9600 Annandale, VA. Fidonet 1:109/305
Rem-Jem, 703-503-9410. 9600 Fairfax, VA.
Enlightend, 703-370-9528. 14.4k Alexandria, VA. Fidonet 1:109/615
My UnKnown BBS, 703-690-0669. 14.4k V.32bis VA. Fidonet 1:109/370
Georgia Peach BBS, 804-727-0399. 14.4k Newport News, VA.
S'Qually Holler, 206-235-0270. 14.4k USR D/S Renton, WA. FidoNet: 1:343/34, `squally.halcyon.com`, UUCP
Top Hat BBS, 206-244-9661. 14.4k WA. Fidonet 1:343/40
victrola.sea.wa.us, 206-838-7456. 19.2k Federal Way, WA. USENET

Outside of the United States

Galaktische Archive, 0043-2228303804. 16.8 ZYX Wien, Austria. Fidonet 2:310/77 (19:00-7:00)
Linux-Support-Oz, +61-2-418-8750. v.32bis 14.4k Sydney, NSW, Austrailia. Internet/Usenet, E-Mail/News
500cc Formula 1 BBS, +61-2-550-4317. V.32bis Sydney, NSW, Australia.
Magic BBS, 403-569-2882. 14.4k HST/Telebit/MNP Calgary, AB, Canada. Internet/Usenet
Logical Solutions, 299-9900 through 9911. 2400 AB, Canada.
Logical Solutions, 299-9912, 299-9913. 14.4k Canada.
Logical Solutions, 299-9914 through 9917. 16.8k v.32bis Canada.
V.A.L.I.S., 403-478-1281. 14.4k v.32bis Edmonton, AB, Canada. USENET
The Windsor Download, (519)-973-9330. v32bis 14.4 ON, Canada.
r-node, 416-249-5366. 2400 Toronto, ON, Canada. USENET
Synapse, 819-246-2344. 819-561-5268 Gatineau, QC, Canada. RIME->SYNAPSE
Radio Free Nyongwa, 514-524-0829. v.32bis ZyXEL Montreal, QC, Canada. USENET, Fidonet
DataComm1, +49.531.132-16. 14.4 HST Braunschweig, NDS, Germany. Fido 2:240/550, LinuxNet
DataComm2, +49.531.132-17. 14.4 HST Braunschweig, NDS, Germany. Fido 2:240/551, LinuxNet
Linux Server /Braukmann, +49.441.592-963. 16.8 ZYX Oldenburg, NDS, Germany. Fido 2:241/2012, LinuxNet
MM's Spielebox, +49.5323.3515. 14.4 ZYX Clausthal-Zfd., NDS, Germany. Fido 2:241/3420
MM's Spielebox, +49.5323.3516. 16.8 ZYX Clausthal-Zfd., NDS, Germany. Fido 2:241/3421

MM's Spielebox, +49.5323.3540. 9600 Clausthal-Zfd., NDS, Germany. Fido 2:241/3422
Bit-Company / J. Bartz, +49.5323.2539. 16.8 ZYX MO Clausthal-Zfd., NDS, Germany. Fido 2:241/3430
Fractal Zone BBS /Maass, +49.721.863-066. 16.8 ZYX Karlsruhe, BW, Germany. Fido 2:241/7462
Hipposoft /M. Junius, +49.241.875-090. 14.4 HST Aachen, NRW, Germany. Fido 2:242/6, 4:30-7,8-23:30
UB-HOFF /A. Hoffmann, +49.203.584-155. 19.2 ZYX+ Duisburg, Germany. Fido 2:242/37
FORMEL-Box, +49.4191.2846. 16.8 ZYX Kaltenkirchen, SHL, Germany. Fido 2:242/329, LinuxNet (6:00-20:00)
BOX/2, +49.89.601-96-77. 16.8 ZYX Muenchen, BAY, Germany. Fido 2:246/147, info magic: LINUX (22-24,0:30-2,5-8)
Die Box Passau 2+1, +49.851.555-96. 14.4 V32b Passau, BAY, Germany. Fido 2:246/200 (8:00-3:30)
Die Box Passau Line 1, +49.851.753-789. 16.8 ZYX Passau, BAY, Germany. Fido 2:246/2000 (8:00-3:30)
Die Box Passau Line 3, +49.851.732-73. 14.4 HST Passau, BAY, Germany. Fido 2:246/202 (5:00-3:30)
Die Box Passau ISDN, +49.851.950-464. 38.4/64k V.110/X.75 Passau, BAY, Germany. Fido 2:246/201 (8:00-24:00,1:00-3:30)
Public Domain Kiste, +49.30.686-62-50. 16.8 ZYX BLN, Germany. Fido 2:2403/17
CS-Port / C. Schmidt, +49.30.491-34-18. 19.2 Z19 Berlin, BLN, Germany. Fido 2:2403/13
BigBrother / R. Gmelch, +49.30.335-63-28. 16.8 Z16 Berlin, BLN, Germany. Fido 2:2403/36.4 (16-23:00)
CRYSTAL BBS, +49.7152.240-86. 14.4 HST Leonberg, BW, Germany. Fido 2:2407/3, LinuxNet
Echoblaster BBS #1, +49.7142.213-92. HST/V32b Bietigheim, BW, Germany. Fido 2:2407/4, LinuxNet (7-19,23-01h
Echoblaster BBS #2, +49.7142.212-35. V32b Bietigheim, BW, Germany. Fido 2:2407/40, LinuxNet (20h-6h)
LinuxServer / P. Berger, +49.711.756-275. 16.8 HST Stuttgart, BW, Germany. Fido 2:2407/34, LinuxNet (8:3-17:5,19-2)
Rising Sun BBS, +49.7147.3845. 16.8 ZYX Sachsenheim, BW, Germany. Fido 2:2407/41, LinuxNet (5:30-2:30)
bakunin.north.de, +49.421.870-532. 14.4 D 2800 Bremen, HB, Germany. `kraehe@bakunin.north.de`
oytix.north.de, +49.421.396-57-62. ZYX HB, Germany. `mike@oytix.north.de`, login as `gast`
Fiffis Inn BBS, +49-89-5701353. 14.4-19.2 Munich, Germany. FidoNet 2:246/69,Internet,USENET,LinuxNet
The Field of Inverse Chaos, +358 0 506 1836. 14.4k v32bis/HST Helsinki, Finland. USENET; ichaos.nullnet.fi
Modula BBS, +33-1 4043 0124. HST 14.4 v.32bis Paris, France.
Modula BBS, +33-1 4530 1248. HST 14.4 V.32bis Paris, France.
STDIN BBS, +33-72375139. v.32bis Lyon, Laurent Cas, France. FidoNet 2:323/8
Le Lien, +33-72089879. HST 14.4/V32bis Lyon, Pascal Valette, France. FidoNet 2:323/5

Basil, +33-1-44670844. v.32bis Paris, Laurent Chemla, France.
Cafard Naum, +33-51701632. v.32bis Nantes, Yann Dupont, France.
DUBBS, +353-1-6789000. 19.2 ZyXEL Dublin, Ireland. Fidonet 2:263/167
Galway Online, +353-91-27454. 14.4k v32b Galway, Ireland. RIME, `@iol.ie`
Nemesis' Dungeon, +353-1-324755 or 326900. 14.4k v32bis Dublin, Ireland. Fidonet 2:263/150
nonsolosoftware, +39 51 6140772. v.32bis, v.42bis Italy. Fidonet 2:332/407
nonsolosoftware, +39 51 432904. ZyXEL 19.2k Italy. Fidonet 2:332/417
Advanced Systems, +64-9-379-3365. ZyXEL 16.8k Auckland, New Zealand. Singet, INTLnet, Fidonet
Thunderball Cave, 472567018. Norway. RIME ->CAVE
DownTown BBS Lelystad, +31-3200-48852. 14.4k Lelystad, Netherlands. Fido 2:512/155, UUCP
MUGNET Intl-Cistron BBS, +31-1720-42580. 38.4k Alphen a/d Rijn, Netherlands. UUCP
The Controversy, (65)560-6040. 14.4k V.32bis/HST Singapore. Fidonet 6:600/201
Pats System, +27-12-333-2049. 14.4k v.32bis/HST Pretoria, South Africa. Fidonet 5:71-1/36
Gunship BBS, +46-31-693306. 14.4k HST DS Gothenburg Sweden.
Baboon BBS, +41-62-511726. 19.2k Switzerland. Fido 2:301/580 and /581
The Purple Tentacle, +44-734-590990. HST/V32bis Reading, UK. Fidonet 2:252/305
A6 BBS, +44-582-460273. 14.4k Herts, UK. Fidonet 2:440/111
On the Beach, +444-273-600996. 14.4k/16.8k Brighton, UK. Fidonet 2:441/122

Appendix E

The GNU General Public License

Printed below is the GNU General Public License (the *GPL* or *copyleft*), under which Linux is licensed. It is reproduced here to clear up some of the confusion about Linux's copyright status—Linux is *not* shareware, and it is *not* in the public domain. The bulk of the Linux kernel is copyright ©1993 by Linus Torvalds, and other software and parts of the kernel are copyrighted by their authors. Thus, Linux *is* copyrighted, however, you may redistribute it under the terms of the GPL printed below.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

E.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights

or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

E.2 Terms and Conditions for Copying, Distribution, and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE

DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

E.3 Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> ⟨*one line to give the program's name and a brief idea of what it does.*⟩ Copyright ©19yy ⟨*name of author*⟩
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with
ABSOLUTELY NO WARRANTY; for details type 'show w'.  This is free software, and
you are welcome to redistribute it under certain conditions; type 'show c' for
details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> ⟨*signature of Ty Coon*⟩, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.