Linux Installation and Getting Started

1992–1998 Matt Welsh
Phil Hughes
David Bandel
Boris Beletsky
Sean Dreilinger
Robert Kiesling
Evan Liebovitch
Henry Pierce

Version 3.2, 20 February 1998.

This book is meant for UNIX novices and gurus alike. It contains information on how to obtain Linux, software installation, a tutorial for new Linux users, and an introduction to system administration. It is meant to be general enough to be applicable to any distribution of Linux.

You may freely copy and redistribute this book under certain conditions. Please see the copyright and distribution statement.

Names of all products herein are used for identification purposes only and are trademarks and/or registered trademarks of their respective owners. Specialized Systems Consultants, Inc., makes no claim of ownership or corporate association with the products or companies that own them.
Copyright ©1992-1996 Matt Welsh
Copyright ©1998 Specialized Systems Consultants, Inc (SSC)

P.O. Box 55549
Seattle, WA 98155-0549
USA
Phone: +1-206-782-7733
Fax: +1-206-782-7191
E-mail: `ligs@ssc.com`
URL: `http://www.ssc.com/`

Linux Installation and Getting Started is a free document; you may reproduce and/or modify it under the terms of version 2 (or, at your option, any later version) of the GNU General Public License as published by the Free Software Foundation.

This book is distributed in the hope 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, in Appendix C.

The authors encourage wide distribution of this book for personal or commercial use, provided the above copyright notice remains intact and the method adheres to the provisions of the GNU General Public License (see Appendix C). In summary, you may copy and distribute this book free of charge or for a profit. No explicit permission is required from the author for reproduction of this book in any medium, physical or electronic.

Note, derivative works and translations of this document *must* be placed under the GNU General Public License, and the original copyright notice must remain intact. If you have contributed new material to this book, you must make the source code (e.g., LaTeX source) available for your revisions. Please make revisions and updates available directly to the document maintainers, Specialized Systems Consultants. This will allow for the merging of updates and provide consistent revisions to the Linux community.

If you plan to publish and distribute this book commercially, donations, royalties, and/or printed copies are greatly appreciated by the authors and the Linux Documentation Project. Contributing in this way shows your support for free software and the Linux Documentation Project. If you have questions or comments, please contact SSC.

Contents

Preface **ix**

Hints for UNIX novices. x
Hints for UNIX gurus. xi
Audience. xii
Organization. xii
Acknowledgments. xiii
Cedits and legalese. xiv
Conventions. xv

1 Introduction to Linux **1**

1.1 About this book. 2
1.2 A brief history of Linux. 3
1.3 System features. 5
1.4 Software features. 6
1.4.1 Text processing and word processing. 8
1.4.2 Programming languages and utilities. 11
1.4.3 Introduction to the X Window System. 12
1.4.4 Introduction to Networking. 13
1.4.5 Telecommunications and BBS software. 14
1.4.6 World Wide Web. 15
1.4.7 Interfacing and MS-DOS. 16
1.4.8 Other applications. 17
1.5 Copyright issues. 18
1.6 The design and philosophy of Linux. 19
1.7 Differences between Linux and other operating systems. 21
1.8 Hardware requirements. 25

1.9 Sources of Linux information. 29
1.9.1 Online documents. 29
1.9.2 Linux on the World Wide Web. 30
1.9.3 Books and other published works. 30
1.9.4 Usenet newsgroups. 31
1.9.5 Internet mailing lists. 33
1.10 Getting Help with Linux. 33

2 Obtaining and Installing Linux 36
2.1 Generic installation. 36
2.1.1 Major Linux distributions. 37
2.1.2 Common concerns. 37
2.1.3 Hardware. 38
2.1.4 Planning. 39
2.1.5 System planning worksheet. 39
2.1.6 Mice. 40
2.1.7 Considering Hard drives and CD-ROMs. 41
2.1.8 Disk drives under Linux. 42
2.1.9 Installing The X Window System 42
2.1.10 Networking hardware. 43
2.1.11 Planning, Part 2. 44
2.1.12 Partitioning strategies. 44
2.1.13 The swap partition. 47
2.1.14 Repartitioning. 47
2.1.15 Backing up your old system. 48
2.1.16 `FIPS.EXE` 49
2.1.17 Preparing to boot Linux. 50
2.1.18 Creating a Linux boot disk under DOS. 50
2.1.19 Creating a Linux boot disk under Linux. 50
2.1.20 Partitioning the hard disk: `fdisk` and `cfdisk`. 51
2.2 Linux distributions. 53
2.3 Debian GNU/Linux. 54
2.3.1 Debian GNU/Linux installation features. 54
2.3.2 Getting floppy images. 54
2.3.3 Downloading the packages. 56
2.3.4 Booting from floppies and installing Debian GNU/Linux. 56

2.3.5 Running Debian GNU/Linux. 64
2.3.6 `dselect`. 67
2.3.7 `dpkg`. 69
2.3.8 About Debian GNU/Linux. 71
2.3.9 Mailing lists. 71
2.3.10 Bug tracking system. 72
2.3.11 Debian Acknowledgments. 72
2.3.12 Last note. 73
2.4 Red Hat Linux. 73
2.4.1 Red Hat Linux installation features. 73
2.4.2 The RPM package management system. 73
2.4.3 A note about upgrading Red Hat Linux. 74
2.4.4 Creating the installation floppies. 75
2.4.5 Installation media. 76
2.4.6 Customizing your NFS or hard drive installation. 77
2.4.7 Recommended minimal installation. 79
2.4.8 How much space do you really need? 80
2.4.9 Installation. 80
2.4.10 Installation media revisited. 80
2.4.11 Walking through the rest of the installation. 82
2.4.12 After installation. 84
2.5 Caldera OpenLinux 85
2.5.1 Obtaining Caldera OpenLinux. 85
2.5.2 Preparing to install Caldera OpenLinux. 86
2.5.3 Creating boot/modules floppies. 86
2.5.4 Preparing the hard disks. 87
2.6 Slackware 87
2.6.1 Slackware is not for you. (Or maybe it is.) 88
2.6.2 A quick history. 88
2.6.3 Why, then? 89
2.6.4 Upgrade? Think twice! 89
2.6.5 Select an installation method. 90
2.6.6 Boot disks: always a good thing. 92
2.6.7 Slackware setup worksheet. 92
2.6.8 Making Slackware happen. 102
2.6.9 Build some boot disks. 103

2.6.10 Boot into action. . . . 103
2.6.11 The Slackware `setup` program. . . . 103
2.6.12 Is that all? . . . 104
2.6.13 Troubleshooting difficult deliveries. . . . 105
2.6.14 Basking in the afterglow. . . . 106
2.6.15 Consider reinstalling! . . . 106
2.6.16 Secure the system. . . . 107
2.7 S.u.S.E. . . . 109
2.7.1 Beginning the installation. . . . 109
2.7.2 S.u.S.E Post-installation. . . . 110
2.7.3 Getting X up and running. . . . 111
2.7.4 Later upgrades. . . . 112
2.8 Post-installation procedures. . . . 112
2.9 Running into trouble. . . . 113
2.9.1 Problems with booting the installation media . . . 113
2.9.2 Hardware problems. . . . 116
2.9.3 Problems installing the software. . . . 120
2.9.4 Problems after installing Linux. . . . 122

3 Linux Tutorial 131
3.1 Introduction. . . . 131
3.2 Basic Linux concepts. . . . 131
3.2.1 Creating an account. . . . 132
3.2.2 Logging in. . . . 132
3.2.3 Virtual consoles. . . . 133
3.2.4 Shells and commands. . . . 133
3.2.5 Logging out. . . . 135
3.2.6 Changing your password. . . . 135
3.2.7 Files and directories. . . . 135
3.2.8 The directory tree. . . . 136
3.2.9 The current working directory. . . . 137
3.2.10 Referring to home directories. . . . 138
3.3 First steps into Linux. . . . 138
3.3.1 Moving around. . . . 139
3.3.2 Looking at the contents of directories. . . . 140
3.3.3 Creating new directories. . . . 142

3.3.4 Copying files. . . . 142
3.3.5 Moving files. . . . 143
3.3.6 Deleting files and directories. . . . 143
3.3.7 Looking at files. . . . 143
3.3.8 Getting online help. . . . 144
3.4 Accessing MS-DOS files. . . . 145
3.5 Summary of basic UNIX commands. . . . 146
3.6 Exploring the file system. . . . 148
3.7 Types of shells. . . . 153
3.8 Wildcards. . . . 154
3.9 Linux plumbing. . . . 157
3.9.1 Standard input and standard output. . . . 157
3.9.2 Redirecting input and output. . . . 158
3.9.3 Using pipes. . . . 159
3.9.4 Non-destructive redirection of output. . . . 161
3.10 File permissions. . . . 161
3.10.1 Concepts of file permissions. . . . 161
3.10.2 Interpreting file permissions. . . . 162
3.10.3 Permissions Dependencies. . . . 163
3.10.4 Changing permissions. . . . 164
3.11 Managing file links. . . . 164
3.11.1 Hard links. . . . 164
3.11.2 Symbolic links. . . . 166
3.12 Job control. . . . 166
3.12.1 Jobs and processes. . . . 166
3.12.2 Foreground and background. . . . 167
3.12.3 Backgrounding and killing jobs. . . . 168
3.12.4 Stopping and restarting jobs. . . . 170
3.13 Using the `vi` editor. . . . 172
3.13.1 Concepts. . . . 173
3.13.2 Starting `vi`. . . . 173
3.13.3 Inserting text. . . . 174
3.13.4 Deleting text. . . . 175
3.13.5 Changing text. . . . 176
3.13.6 Commands for moving the cursor. . . . 177
3.13.7 Saving files and quitting `vi`. . . . 178

3.13.8 Editing another file. . . . 178
3.13.9 Including other files. . . . 179
3.13.10 Running shell commands. . . . 179
3.13.11 Getting `vi` help. . . . 180
3.14 Customizing your environment. . . . 180
3.14.1 Shell scripts. . . . 181
3.14.2 Shell variables and the environment. . . . 182
3.14.3 Shell initialization scripts. . . . 185
3.15 So you want to strike out on your own? . . . 186

4 System Administration 187
4.1 The `root` account. . . . 187
4.2 Booting the system. . . . 188
4.2.1 Using LILO. . . . 189
4.3 Shutting down. . . . 191
4.3.1 The `/etc/inittab` file. . . . 192
4.4 Managing file systems. . . . 196
4.4.1 Mounting file systems. . . . 196
4.4.2 Device driver names. . . . 198
4.4.3 Checking file systems. . . . 199
4.5 Using a swap file. . . . 200
4.6 Managing users. . . . 201
4.6.1 User management concepts. . . . 201
4.6.2 Adding users. . . . 202
4.6.3 Deleting users. . . . 204
4.6.4 Setting user attributes. . . . 205
4.6.5 Groups. . . . 205
4.6.6 System administration responsibilities. . . . 206
4.6.7 Coping with users. . . . 206
4.6.8 Setting the rules. . . . 207
4.6.9 What it all means. . . . 207
4.7 Archiving and compressing files. . . . 208
4.7.1 Using `tar`. . . . 208
4.7.2 `gzip` and `compress`. . . . 209
4.7.3 Putting them together. . . . 210
4.8 Using floppies and making backups. . . . 211

4.8.1 Using floppies for backups. . . . 212
4.8.2 Backups with a Zip drive. . . . 213
4.8.3 Making backups to tape devices. . . . 214
4.8.4 Using floppies as file systems. . . . 215
4.9 Upgrading and installing new software. . . . 216
4.9.1 Upgrading the kernel 217
4.9.2 Adding a device driver to the kernel. . . . 219
4.9.3 Installing a device driver module. . . . 221
4.9.4 Upgrading the libraries. . . . 223
4.9.5 Upgrading `gcc`. . . . 224
4.9.6 Upgrading other software. . . . 224
4.10 Miscellaneous tasks. . . . 224
4.10.1 System startup files. . . . 225
4.10.2 Setting the host name. . . . 225
4.11 What to do in an emergency. . . . 226
4.11.1 Recovery with a maintenance diskette. . . . 227
4.11.2 Fixing the root password. . . . 227
4.11.3 Trashed file systems. . . . 228
4.11.4 Recovering lost files. . . . 228
4.11.5 Trashed libraries. . . . 228

5 The X Window System 229
5.1 X Window Hardware requirements. . . . 230
5.1.1 Video display. . . . 230
5.1.2 Memory, CPU, and disk space. . . . 231
5.2 XFree86 installation. . . . 231
5.3 Probing the hardware configuration. . . . 234
5.4 Automatically generating the `XF86Config` file. . . . 235
5.5 Configuring XFree86. . . . 235
5.6 Filling in video card information. . . . 244
5.7 Running XFree86. . . . 247
5.8 When you run into trouble. . . . 248

6 Networking 250
6.1 Networking with TCP/IP. . . . 250
6.1.1 Configuring TCP/IP on your system. . . . 251
6.1.2 SLIP configuration. . . . 260

6.2 Dial-up networking and PPP. 266
6.2.1 What you need to get started. 266
6.2.2 An overview of the steps involved. 267
6.2.3 Creating the connection scripts. 279
6.2.4 Editing the supplied PPP startup scripts. 282
6.2.5 Starting PPP at the server end. 284
6.2.6 If your PPP server uses PAP (Password Authentication Protocol). . 285
6.2.7 Using MSCHAP. 286
6.2.8 Shutting down the PPP link. 289
6.2.9 Troubleshooting common problems once the link is working. 289
6.3 Networking with UUCP. 291
6.4 Networking with Microsoft Systems. 292
6.5 Electronic mail. 292
6.6 News and Usenet. 293

A Sources of Linux Information 296

B FTP Tutorial and Site List 310

C The GNU General Public License 318

Preface

Linux Installation and Getting Started (LIGS) has been the shepherding work for countless new users of the Linux operating system. Linux continues to evolve and so, too, must this guide.

Matt Welsh, the original author, has turned the book over to the care and management of Specialized Systems Consultants, Inc. (SSC), publishers of *Linux Journal*, computer books, and references. *Linux Installation and Getting Started* is still covered by the GNU General Public License—it is still freely redistributable, like the operating system it describes. This new version becomes a collaborative effort of individuals separated by geography but brought together on the Internet, much like Linux itself. If you believe you could expand or update a section of *Linux Installation and Getting Started* or have something new and wonderful to add, please send e-mail to `ligs@ssc.com` and tell us how you'd like to contribute.

For this edition, we've added distribution-specific instructions for obtaining and installing S.u.S.E. Linux, Debian GNU/Linux, Linux Slackware, Caldera OpenLinux, and Red Hat Linux. Please read through the acknowledgements, and if you should meet someone named there on line or in person, thank them for the help.

Specialized Systems Consultants, Inc. (SSC)
February 1998

Preface to the previous edition.

"You are in a maze of twisty little passages, all alike."

Before you looms one of the most complex and utterly intimidating systems ever written. Linux, the free UNIX clone for the personal computer, produced by a mishmash team of UNIX gurus, hackers, and the occasional loon. The system itself reflects this complex

heritage, and although the development of Linux may appear to be a disorganized volunteer effort, the system is powerful, fast, and free. It is a true 32-bit operating system solution.

My own experiences with Linux began several years ago when I sat down to figure out how to install the only "distribution" available at the time—a couple of diskettes made available by H. J. Lu. I downloaded a slew of files and read page upon page of loosely-organized installation notes. Somehow, I managed to install this basic system and get everything to work together. This was long before you could buy the Linux software on CD-ROM from worldwide distributors; before, in fact, Linux was able to access a CD-ROM drive. This was before XFree86, before Emacs, before commercial software support, and before Linux became a true rival to MS-DOS, Microsoft Windows, and OS/2 in the personal computer market.

You hold in your hands a map and guidebook to the world of Linux. It is my hope that this book will help you get rolling with what I consider to be the fastest, most powerful operating system for the personal computer. Setting up your own Linux system can be great fun—so grab a cup of coffee, sit back, and read on.

Matt Welsh
January 1994

Hints for UNIX novices.

Getting started with your own Linux system does not require a great deal of UNIX background. Many UNIX novices have successfully installed Linux on their systems. This is a worthwhile learning experience, but keep in mind that it can be frustrating. Moreover, once you are ready to delve into the more complex tasks of running Linux—installing new software, recompiling the kernel, and so forth—having background knowledge in UNIX is necessary.

However, simply by running your own Linux system you will learn the essentials of UNIX. This book helps you get started—Chapter 3 is a tutorial covering UNIX basics. Chapter 4 has information on Linux system administration. You may wish to read these chapters before attempting to install Linux at all—the information will prove to be invaluable should you run into problems.

Nobody can expect to go from UNIX novice to UNIX system administrator overnight. No implementation of UNIX is expected to be maintenance free. You must be prepared for the journey that lies ahead. Otherwise, if you're new to UNIX, you may very well become frustrated with the system.

Hints for UNIX gurus.

Someone with years of experience in UNIX programming and system administration may still need assistance before he or she is able to pick up and install Linux. UNIX wizards must be familiar with certain aspects of the system before they dive in. Linux is neither a commercial UNIX system, nor attempts to uphold the same standards. While stability is an important factor in Linux development, it is not the only factor.

Perhaps more important is functionality. In many cases, new code becomes part of the standard kernel while it is still buggy and not functionally complete. The Linux development model assumes that it is more important to release code for users to test and use, than delay a release until it is complete. WINE (the Microsoft Windows Emulator for Linux) had an official alpha release before it was completely tested. The Linux community at large had a chance to work with the code, and those who found the alpha code good enough for their needs could use it. Commercial UNIX vendors rarely, if ever, release software this way.

If you have been a UNIX systems administrator for more than a decade, and have used every commercial UNIX system under the Sun (pun intended), Linux may take some getting used to. The system is very modern and dynamic. A new kernel is released every few weeks. New software is constantly being released. One day, your system may be completely up-to-date, and the next day the system may be in the Stone Age.

With all of this activity, how does one keep up with the ever-changing Linux world? For the most part, it is best to upgrade only those parts of the system which need upgrading, and only when you think it is necessary. For example, if you never use Emacs, there is little reason to continuously install new releases of Emacs on your system. Furthermore, even if you are an avid Emacs user, there is usually no reason to upgrade unless you need a feature that is present only in the next release. There is little or no reason to always be on top of the newest software versions.

We hope that Linux will meet or exceed your expectations for a homebrew UNIX system. At the very core of Linux is the spirit of free software, of constant development and growth. The Linux community favors expansion over stability, which is a difficult concept to swallow, especially after being steeped in the world of commercial UNIX. Expecting Linux to be perfect is unrealistic; nothing in the free software world ever is. We believe, however, that Linux is as complete and useful as any other implementation of UNIX.

Audience.

This book is for personal computer users who want to install and use Linux. We assume that you have basic knowledge about personal computers and operating systems like MS-DOS, but no previous knowledge of Linux or UNIX.

Despite this, we strongly suggest that UNIX novices invest in one of the many good UNIX books out there. You still need UNIX know-how to install and run a complete system. No distribution of Linux is completely bug-free. You may be required to fix small problems by hand. Running a UNIX system is not an easy task, even with commercial versions of UNIX. If you're serious about Linux, bear in mind that it takes considerable effort and attention to keep the system running. This is true of any UNIX system. Because of the diversity of the Linux community and the many needs which the software attempts to meet, not everything can be taken care of for you all of the time.

Organization.

This book contains the following chapters:

Chapter 1, *Introduction to Linux*, is a general introduction to Linux, its capabilities, and requirements for running it on your system. It also provides hints for getting help and reducing your stress level.

Chapter 2, *Obtaining and Installing Linux*, explains how to obtain and install Linux software, beginning with drive repartitioning, creating filesystems, and installing software packages. The chapter contains instructions that are meant to be general for any Linux distribution and relies for specifics on the documentation provided by your particular release.

Chapter 3, *Linux Tutorial*, is a complete introduction for UNIX novices. If you have previous UNIX experience, most of this material should be familiar.

Chapter 4, *System Administration*, introduces important concepts for system administration under Linux. This will be of interest to UNIX system administrators who want to know about the Linux-specific issues for running a system.

Chapters 5 and 6, *X Windows* and *Networking*, introduce a number of advanced options that Linux supports, like the X Window System and TCP/IP networking. We also provide a complete guide to configuring XFree86-3.1.

Appendix A, *Sources of Linux Information*, is a list of further documentation sources like newsgroups, mailing lists, on-line documents, and books.

Appendix B, *FTP Tutorial and Site List*, is a tutorial for downloading files from the Internet with FTP. This appendix also lists FTP archive sites that carry Linux software.

Appendix C, *The GNU General Public License*, is the license agreement under which Linux is distributed. It is important that Linux users understand the GPL. Many disagreements over the terms in describes have been raised.

Acknowledgments.

This edition builds on the work of those who have gone before, and they are thanked below in Matt Welsh's original acknowledgement. Additionally, we owe thanks to Larry Ayers, Boris Beletsky, Sean Dreilinger, Evan Leibovitch, and Henry Pierce for contributing the information in Chapter 2 on S.u.S.E. Linux, Debian GNU/Linux, Linux Slackware, Caldera OpenLinux, and Red Hat Linux, respectively. David Bandel updated Chapter 2 and added a section describing a generic Linux installation. Vernard Martin updated and added to Chapter 5. Thanks are also due to Belinda Frazier for editing and to Jay Painter for the update to Chapter 4 on systems administration.

Acknowledgments from the previous edition.

This book has been long in the making, and many people have contributed to the outcome. In particular, I would like to thank Larry Greenfield and Karl Fogel for their work on the first version of Chapter 3, and to Lars Wirzenius for his work on Chapter 4. Thanks to Michael K. Johnson for his assistance with the LDP and the LaTeX conventions used in this manual, and to Ed Chi, who sent me a printed copy of the book.

Thanks to Melinda A. McBride at SSC, Inc., who did an excellent job of completing the index for Chapters 3, 4, and **??**. I would also like to thank Andy Oram, Lar Kaufman, and Bill Hahn at O'Reilly and Associates for their assistance with the Linux Documentation Project.

Thanks to Linux Systems Labs, Morse Telecommunications, and Yggdrasil Computing for their support of the Linux Documentation Project through sales of this book and other works.

Much thanks to the many Linux activists, including (in no particular order) Linus Torvalds, Donald Becker, Alan Cox, Remy Card, Ted T'so, H. J. Lu, Ross Biro, Drew Eckhardt, Ed Carp, Eric Youngdale, Fred van Kempen, and Steven Tweedie, for devoting so much time and energy to this project, and without whom there wouldn't be anything to write a book about.

Finally, special thanks to the myriad of readers who have sent their helpful comments and corrections; they are far too many to list here.

Credits and legalese.

The Linux Documentation Project consists of a loose team of writers, proofreaders, and editors who are working on a set of definitive Linux manuals.

This manual is one of several which are distributed by the Linux Documentation Project. Other manuals include the *Linux User's Guide*, *System Administrator's Guide*, *Network Administrator's Guide*, and *Kernel Hacker's Guide.* These manuals are all available in LaTeX source and PostScript output format for anonymous FTP access at `sunsite.unc.edu`, in the directory `/pub/Linux/docs/LDP`.

Conventions.

We have attempted to use the following documentation conventions in this guide:

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 beginnings of sections.

slanted — Used to mark **meta-variables** in the text, especially in command lines. For example, in

```
ls -l foo
```

foo represents a file name, 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 C code, shell scripts, or to display files like configuration files. When necessary for the sake of clarity, these examples or figures are enclosed in thin boxes.

`Key` — Represents a key to press, such as in this example

Press `Enter` to continue.

◇ — A diamond in the margin, like a black diamond on a ski hill, marks “danger” or “caution”. Carefully read the paragraphs so marked.

Chapter 1

Introduction to Linux

Linux is quite possibly the most important free software achievement since the original Space War, or, more recently, Emacs. It has developed into an operating system for business, education, and personal productivity. Linux is no longer only for UNIX wizards who sit for hours in front of a glowing console (although we assure you that many users fall into this category). This book will help you get the most from Linux.

Linux (pronounced with a short *i*, as in *LIH-nucks*) is a UNIX operating system clone which runs on a variety of platforms, especially personal computers with Intel 80386 or better processors. It supports a wide range of software, from TEX, to the X Window System, to the GNU C/C++ compiler, to TCP/IP. It's a versatile, bona fide implementation of UNIX, freely distributed under the terms of the GNU General Public License (see Appendix C).

Linux can turn any 80386 or better personal computer into a workstation that puts the full power of UNIX at your fingertips. Businesses install Linux on entire networks of machines, and use the operating system to manage financial and hospital records, distributed computing environments, and telecommunications. Universities worldwide use Linux to teach courses on operating system programming and design. Computing enthusiasts everywhere use Linux at home for programming, productivity, and all-around hacking.

What makes Linux so different is that it is a free implementation of UNIX. It was and still is developed cooperatively by a group of volunteers, primarily on the Internet, who exchange code, report bugs, and fix problems in an open-ended environment. Anyone is welcome to join the Linux development effort. All it takes is interest in hacking a free UNIX clone, and some programming know-how. The book in your hands is your tour guide.

1.1 About this book.

This book is an installation and entry-level guide to Linux. The purpose is to get new users up and running by consolidating as much important material as possible into one book. Instead of covering volatile technical details which tend to change with rapid development, we give you the straight background to find out more on your own.

Linux is not difficult to install and use. However, as with any implementation of UNIX, there is often black magic involved to get everything working correctly. We hope that this book will get you on the Linux tour bus and show you how great an operating system can be.

In this book, we cover the following topics:

- What is Linux? The design and philosophy of this unique operating system, and what it can do for you.
- Details of running Linux, including suggestions on recommended hardware configuration.
- Specific instructions to install various Linux distributions, including Debian, Red Hat Software, and Slackware.
- A brief, introductory UNIX tutorial for users with no previous UNIX experience. This tutorial should provide enough material for novices to find their way around the system.
- An introduction to system administration under Linux. This covers the most important tasks that Linux administrators need to perform, like creating user accounts and managing file systems.
- Information on configuring more advanced features of Linux, like the X Window System, TCP/IP networking, and electronic mail and news.

This book is for the personal computer user who wishes to get started with Linux. We don't assume previous UNIX experience but do expect novices to refer to other material along the way. For those unfamiliar with UNIX, a list of useful references is given in Appendix A. In general, this book is meant to be read in addition to another book on basic UNIX concepts.

1.2 A brief history of Linux.

UNIX is one of the most popular operating systems worldwide because of its large support base and distribution. It was originally developed at AT&T as a multitasking system for minicomputers and mainframes in the 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.

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.

Versions of UNIX exist for many systems, from personal computers to supercomputers like the Cray Y-MP. Most versions of UNIX for personal computers are expensive and cumbersome. At the time of this writing, a one-machine version of UNIX System V for the 386 runs about US$1500.

Linux is a free version of UNIX developed primarily by Linus Torvalds at the University of Helsinki in Finland, with the help of many UNIX programmers and wizards across the Internet. Anyone with enough know-how and gumption can 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 was developed by the GNU project of the Free Software Foundation in Cambridge, Massachusetts, U.S.A. However, programmers from all over the world have contributed to the growing pool of Linux software.

Linux was originally developed as a hobby project by Linus Torvalds. It was inspired by Minix, a small UNIX system developed by Andy Tanenbaum. The first discussions about Linux were on the Usenet newsgroup, `comp.os.minix`. These discussions were concerned mostly with the development of a small, academic UNIX system for Minix users who wanted more.

The very early development of Linux mostly dealt with the task-switching features of the 80386 protected-mode interface, all written in assembly code. Linus writes,

> "After that it was plain sailing: hairy coding still, but I had some devices, and debugging was easier. I started using C at this stage, and it certainly speeds up development. This is also when I started to get serious about my megalomaniac ideas to make 'a better Minix than Minix.' I was hoping I'd be able to recompile `gcc` under Linux someday...
>
> "Two months for basic setup, but then only slightly longer until I had a disk driver (seriously buggy, but it happened to work on my machine) and a small file system. That was about when I made 0.01 available (around late August

> of 1991): it wasn't pretty, it had no floppy driver, and it couldn't do much of anything. I don't think anybody ever compiled that version. But by then I was hooked, and didn't want to stop until I could chuck out Minix."

No announcement was ever made for Linux version 0.01. The 0.01 sources weren't even executable. They contained only the bare rudiments of the kernel source and assumed that you had access to a Minix machine to compile and experiment with them.

On October 5, 1991, Linus announced the first "official" version of Linux, which was version 0.02. At that point, Linus was able to run `bash` (the GNU Bourne Again Shell) and `gcc` (the GNU C compiler), but not much else. Again, this was intended as a hacker's system. The primary focus was kernel development—user support, documentation, and distribution had not yet been addressed. Today, the Linux community still seems to treat these issues as secondary to "real programming"—kernel development.

As Linus wrote in `comp.os.minix`,

> "Do you pine for the nice days of Minix-1.1, when men were men and wrote their own device drivers? Are you without a nice project and just dying to cut your teeth on an OS you can try to modify for your needs? Are you finding it frustrating when everything works on Minix? No more all-nighters to get a nifty program working? Then this post might be just for you.
>
> "As I mentioned a month ago, I'm working on a free version of a Minix-look-alike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be, depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02... but I've successfully run `bash`, `gcc`, `gnu-make`, `gnu-sed`, `compress`, etc. under it."

After version 0.03, Linus bumped up the version number to 0.10, as more people started to work on the system. After several further revisions, Linus increased the version number to 0.95 in March, 1992, to reflect his expectation that the system was ready for an "official" release soon. (Generally, software is not assigned the version number 1.0 until it is theoretically complete or bug-free.). Almost a year and a half later, in late December of 1993, the Linux kernel was still at version 0.99.pl14—asymptotically approaching 1.0. At the time of this writing, the current stable kernel version is 2.0 patchlevel 33, and version 2.1 is under development.

Most of the major, free UNIX software packages have been ported to Linux, and commercial software is also available. More hardware is supported than in the original kernel versions. Many people have executed benchmarks on 80486 Linux systems and found them

comparable with mid-range workstations from Sun Microsystems and Digital Equipment Corporation. Who would have ever guessed that this "little" UNIX clone would have grown up to take on the entire world of personal computing?

1.3 System features.

Linux supports features found in other implementations of UNIX, and many which aren't found elsewhere. In this section, we'll take a nickel tour of the features of the Linux kernel.

Linux is a complete multitasking, multiuser operating system, as are all other versions of UNIX. This means that many users can log into and run programs on the same machine simultaneously.

The Linux system is mostly compatible with several UNIX standards (inasmuch as UNIX has standards) at the source level, including IEEE POSIX.1, UNIX System V, and Berkely System Distribution UNIX. Linux was developed with source code portability in mind, and it's easy to find commonly used features that are shared by more than one platform. Much of the free UNIX software available on the Internet and elsewhere compiles under Linux "right out of the box." In addition, all of the source code for the Linux system, including the kernel, device drivers, libraries, user programs, and development tools, is freely distributable.

Other specific internal features of Linux include POSIX job control (used by shells like `csh` and `bash`), pseudoterminals (`pty` devices), and support for dynamically loadable national or customized keyboard drivers. Linux supports **virtual consoles** that let you switch between login sessions on the same system console. Users of the `screen` program will find the Linux virtual console implementation familiar.

The kernel can emulate 387-FPU instructions, and systems without a math coprocessor can run programs that require floating-point math capability.

Linux supports various file systems for storing data, like the ext2 file system, which was developed specifically for Linux. The Xenix and UNIX System V file systems are also supported, as well as the Microsoft MS-DOS and Windows 95 VFAT file systems on a hard drive or floppy. The ISO 9660 CD-ROM file system is also supported. We'll talk more about file systems in chapters 2 and 4.

Linux provides a complete implementation of TCP/IP networking software. This includes device drivers for many popular Ethernet cards, SLIP (Serial Line Internet Protocol) and PPP (Point-to-Point Protocol), which provide access to a TCP/IP network via a serial connection, PLIP (Parallel Line Internet Protocol), and NFS (Network File System).

The complete range of TCP/IP clients and services is also supported, which includes FTP, `telnet`, NNTP, and SMTP. We'll talk more about networking in Chapter **??**.

The Linux kernel is developed to use protected-mode features of Intel 80386 and better processors. In particular, Linux uses the protected-mode, descriptor based, memory-management paradigm, and other advanced features. Anyone familiar with 80386 protected-mode programming knows that this chip was designed for multitasking systems like UNIX. Linux exploits this functionality.

The kernel supports demand-paged, loaded executables. Only those segments of a program which are actually in use are read into memory from disk. Also, copy-on-write pages are shared among executables. If several instances of a program are running at once, they share physical memory, which reduces overall usage.

In order to increase the amount of available memory, Linux also implements disk paging. Up to one gigabyte of **swap space**[1] may be allocated on disk (upt to 8 partitions of 128 megabytes each). When the system requires more physical memory, it swaps inactive pages to disk, letting you run larger applications and support more users. However, swapping data to disk is no substitute for physical RAM, which is much faster.

The Linux kernel also implements a unified memory pool for user programs and disk cache. All free memory is used by the cache, which is reduced when running large programs.

Executables use dynamically linked, shared libraries: code from a single library on disk. This is not unlike the SunOS shared library mechanism. Executable files occupy less disk space, especially those which use many library functions. There are also statically linked libraries for object debugging and maintaining "complete" binary files when shared libraries are not installed. The libraries are dynamically linked at run time, and the programmer can use his or her own routines in place of the standard library routines.

To facilitate debugging, the kernel generates core dumps for post-mortem analysis. A core dump and an executable linked with debugging support allows a developer to determine what caused a program to crash.

1.4 Software features.

Virtually every utility one would expect of a standard UNIX implementation has been ported to Linux, including basic commands like `ls`, `awk`, `tr`, `sed`, `bc`, and `more`. The familiar working environment of other UNIX systems is duplicated on Linux. All standard

[1] Swap space is inappropriately named; entire processes are not swapped, but rather individual pages. Of course, in many cases, entire processes will be swapped out, but this is not always true.

commands and utilities are included. (Novice UNIX or Linux users should see Chapter 3 for an introduction to basic UNIX commands.)

Many text editors are available, including `vi`, `ex`, `pico`, `jove`, and GNU `emacs`, and variants like Lucid `emacs`, which incorporates extensions of the X Window System, and `joe`. The text editor you're accustomed to using has more than likely been ported to Linux.

The choice of a text editor is an interesting one. Many UNIX users prefer "simple" editors like `vi`. (The original author wrote this book with `vi`.) But `vi` has many limitations due to its age, and modern editors like `emacs` have gained popularity. `emacs` supports a complete, Lisp based macro language and interpreter, powerful command syntax, and other extensions. There are `emacs` macro packages which let you read electronic mail and news, edit directory contents, and even engage in artificially intelligent psychotherapy sessions (indispensible for stressed-out Linux hackers).

Most of the basic Linux utilities are GNU software. GNU utilities support advanced features that are not found in the standard versions of BSD and UNIX System Vprograms. For example, the GNU `vi` clone, `elvis`, includes a structured macro language that differs from the original implementation. However, GNU utilities are intended to remain compatible with their BSD and System V counterparts. Many people consider the GNU versions to be superior to the originals.

A **shell** is a program which reads and executes commands from the user. In addition, many shells provide features like **job control,** managing several processes at once, input and output redirection, and a command language for writing **shell scripts**. A shell script is a program in the shell's command language and is analogous to a MS-DOS batch file.

Many types of shells are available for Linux. The most important difference between shells is the command language. For example, the C SHell (`csh`) uses a command language similar to the C programming language. The classic Bourne SHell `sh` uses another command language. The choice of a shell is often based on the command language it provides, and determines, to a large extent, the qualities of your working environment under Linux.

The GNU Bourne Again Shell (`bash`) is a variation of the Bourne Shell which includes many advanced features like job control, command history, command and filename completion, an `emacs`-like interface for editing command lines, and other powerful extensions to the standard Bourne Shell language. Another popular shell is `tcsh`, a version of the C Shell with advanced functionality similar to that found in `bash`. Other shells include `zsh`, a small Bourne-like shell; the Korn Shell (`ksh`); BSD's `ash`; and `rc`, the Plan 9 shell.

If you're the only person using the system and refer to use `vi` and `bash` exclusively

as your editor and shell, there's no reason to install other editors or shells. This "do it yourself" attitude is prevalent among Linux hackers and users.

1.4.1 Text processing and word processing.

Almost every computer user needs a method of preparing documents. In the world of personal computers, **word processing** is the norm: editing and manipulating text in a "What-You-See-Is-What-You-Get" (WYSIWYG) environment and producing printed copies of the text, complete with graphics, tables, and ornamentation.

Commercial word processors from Corel, Applix, and Star Division are available in the UNIX world, but **text processing,** which is quite different conceptually, is more common. In text processing systems, text is entered in a **page-description language,** which describes how the text should be formatted. Rather than enter text within a special word processing environment, you can modify text with any editor, like `vi` or `emacs`. Once you finish entering the source text (in the typesetting language), a separate program converts the source to a format suitable for printing. This is somewhat analogous to programming in a language like C, and "compiling" the document into printable form.

Many text processing systems are available for Linux. One is `groff`, the GNU version of the classic `troff` text formatter originally developed by Bell Labs and still used on many UNIX systems worldwide. Another modern text processing system is TEX, developed by Donald Knuth of computer science fame. Dialects of TEX, like LATEX, are also available.

Text processors like TEX and `groff` differ mostly in the syntax of their formatting languages. The choice of one formatting system over another is based upon what utilities are available to satisfy your needs, as well as personal taste.

Many people consider `groff`'s formatting language to be a bit obscure and use find TEX more readable. However, `groff` produces ASCII output which can be viewed on a terminal more easily, while TEX is intended primarily for output to a printing device. Various add-on programs are required to produce ASCII output from TEX formatted documents, or convert TEX input to `groff` format.

Another program is `texinfo`, an extension to TEX which is used for software documentation developed by the Free Software Foundation. `texinfo` can produce printed output, or an online-browsable hypertext "Info" document from a single source file. Info files are the main format of documentation used in GNU software like `emacs`.

Text processors are used widely in the computing community for producing papers, theses, magazine articles, and books. (This book is produced using LATEX.) The ability to process source language as a text file opens the door to many extensions of the text

processor itself. Because a source document is not stored in an obscure format that only one word processor can read, programmers can write parsers and translators for the formatting language, and thus extend the system.

What does a formatting language look like? In general, a formatted source file consists mostly of the text itself, with **control codes** to produce effects like font and margin changes, and list formatting.

Consider the following text:

> Mr. Torvalds:
>
> We are very upset with your current plans to implement *post-hypnotic suggestions* in the **Linux** terminal driver code. We feel this way for three reasons:
>
> 1. Planting subliminal messages in the terminal driver is not only immoral, it is a waste of time;
> 2. It has been proven that "post-hypnotic suggestions" are ineffective when used upon unsuspecting UNIX hackers;
> 3. We have already implemented high-voltage electric shocks, as a security measure, in the code for `login`.
>
> We hope you will reconsider.

This text might appear in the LaTeX formatting language as the following:

```
\begin{quote}
Mr. Torvalds:

We are very upset with your current plans to implement
{\em post-hypnotic suggestions\/} in the {\bf Linux} terminal
driver code. We feel this way for three reasons:
\begin{enumerate}
\item Planting subliminal messages in the kernel driver is not only
      immoral, it is a waste of time;
\item It has been proven that ‘‘post-hypnotic suggestions’’
      are ineffective when used upon unsuspecting UNIX hackers;
\item We have already implemented high-voltage electric shocks, as
      a security measure, in the code for {\tt login}.
\end{enumerate}
We hope you will reconsider.
```

```
\end{quote}
```

The author enters the text using any text editor and generates formatted output by processing the source with LaTeX. At first glance, the typesetting language may appear to be obscure, but it's actually quite easy to understand. Using a text processing system enforces typographical standards when writing. All the enumerated lists within a document will look the same, unless the author modifies the definition of an enumerated list. The goal is to allow the author to concentrate on the text, not typesetting conventions.

When writing with a text editor, one generally does not think about how the printed text will appear. The writer learns to visualize the finished text's appearance from the formatting commands in the source.

WYSIWYG word processors are attractive for many reasons. They provide an easy-to-use visual interface for editing documents. But this interface is limited to aspects of text layout which are accessible to the user. For example, many word processors still provide a special format language for producing complicated expressions like mathematical formulae. This is text processing, albeit on a much smaller scale.

A not-so-subtle benefit of text processing is that you specify exactly which format you need. In many cases, the text processing system requires a format specification. Text processing systems also allow source text to be edited with any text editor, instead of relying on format codes which are hidden beneath a word processor's opaque user interface. Further, the source text is easily converted to other formats. The tradeoff for this flexibility and power is the lack of WYSIWYG formatting.

Some programs let you preview the formatted document on a graphics display device before printing. The `xdvi` program displays a "device independent" file generated by the TeX system under X. Applications like `xfig` and `gimp` provide WYSIWYG graphics interfaces for drawing figures and diagrams, which are subsequently converted to text processing language for inclusion in your document.

Text processors like `troff` were around long before WYSIWYG word processing was available. Many people still prefer their versatility and independence from a graphics environment.

Many text-processing-related utilities are available. The powerful METAFONT system, which is used to design fonts for TeX, is included in the Linux port of TeX. Other programs include `ispell`, an interactive spelling checker and corrector; `makeindex`, which generates indices in LaTeX documents; and many other `groff` and TeXbased macro packages which format many types of technical and mathematical texts. Conversion programs that translate between TeX or `groff` source to a myriad of other formats are also available.

A newcomer to text formatting is YODL, written by Karel Kubat. YODL is an easy-to-learn language with filters to produce various output formats, like LaTeX, SGML, and HTML.

1.4.2 Programming languages and utilities.

Linux provides a complete UNIX programming environment which includes all of the standard libraries, programming tools, compilers, and debuggers which you would expect of other UNIX systems.

Standards like POSIX.1 are supported, which allows software written for Linux to be easily ported to other systems. Professional UNIX programmers and system administrators use Linux to develop software at home, then transfer the software to UNIX systems at work. This not only saves a great deal of time and money, but also lets you work in the comfort of your own home. (One of the authors uses his system to develop and test X Window System applications at home, which can be directly compiled on workstations elsewhere.) Computer Science students learn UNIX programming and explore other aspects of the system, like kernel architecture.

With Linux, you have access to the complete set of libraries and programming utilities and the complete kernel and library source code.

Within the UNIX software world, systems and applications are often programmed in C or C++. The standard C and C++ compiler for Linux is GNU `gcc`, which is an advanced, modern compiler that supports C++, including AT&T 3.0 features, as well as Objective-C, another object-oriented dialect of C.

Besides C and C++, other compiled and interpreted programming languages have been ported to Linux, like Smalltalk, FORTRAN, Java, Pascal, LISP, Scheme, and Ada (if you're masochistic enough to program in Ada, we aren't going to stop you). In addition, various assemblers for writing protected-mode 80386 code are available, as are UNIX hacking favorites like Perl (the script language to end all script languages) and Tcl/Tk (a shell-like command processing system which has support for developing simple X Window System applications).

The advanced `gdb` debugger can step through a program one line of source code at a time, or examine a core dump to find the cause of a crash. The `gprof` profiling utility provides performance statistics for your program, telling you where your program spends most of its execution time. As mentioned above, the `emacs` text editor provides interactive editing and compilation environments for various programming languages. Other tools include GNU `make` and `imake`, which manage compilation of large applications, and

RCS, a system for source code locking and revision control.

Finally, Linux supports dynamically linked, shared libraries (DLLs), which result in much smaller binaries. The common subroutine code is linked at run-time. These DLLs let you override function definitions with your own code. For example, if you wish to write your own version of the `malloc()` library routine, the linker will use your new routine instead of the one in the libraries.

1.4.3 Introduction to the X Window System.

The X Window System, or simply X, is a standard graphical user interface (GUI) for UNIX machines and is a powerful environment which supports many applications. Using the X Window System, you can have multiple terminal windows on the screen at once, each having a different login session. A pointing device like a mouse is often used with X, although it isn't required.

Many X-specific applications have been written, including games, graphics and programming utilities, and documentation tools. Linux and X make your system a bona fide workstation. With TCP/IP networking, your Linux machine can display X applications running on other machines.

The X Window System was originally developed at the Massachusetts Institute of Technology and is freely distributable. Many commercial vendors have distributed proprietary enhancements to the original X Window System as well. The version of X for Linux is XFree86, a port of X11R6 which is freely distributable. XFree86 supports a wide range of video hardware, including VGA, Super VGA, and accelerated video adaptors. XFree86 is a complete distribution of the X Windows System software, and contains the X server itself, many applications and utilities, programming libraries, and documents.

Standard X applications include `xterm`, a terminal emulator used for most text-based applications within a window, `xdm`, which handles logins, `xclock`, a simple clock display, `xman`, a X-based manual page reader, and `xmore`. The many X applications available for Linux are too numerous to mention here, but their number includes spreadsheets, word processors, graphics programs, and web browsers like the Netscape Navigator. Many other applications are available separately. Theoretically, any application written for X should compile cleanly under Linux.

The interface of the X Window System is controlled largely by the **window manager**. This user-friendly program is in charge of the placement of windows, the user interface for resizing and moving them, changing windows to icons, and the appearance of window frames, among other tasks. XFree86 includes `twm`, the classic MIT window manager, and

advanced window managers like the Open Look Virtual Window Manager (`olvwm`) are available. Popular among Linux users is `fvwm`—a small window manager that requires less than half the memory of `twm`. It provides a 3-dimensional appearance for windows and a virtual desktop. The user moves the mouse to the edge of the screen, and the desktop shifts as though the display were much larger than it really is. `fvwm` is greatly customizable and allows access to functions from the keyboard as well as mouse. Many Linux distributions use `fvwm` as the standard window manager. A version of `fvwm` called `fvwm95-2` offers Microsoft Windows 95-like look and feel.

The XFree86 distribution includes programming libraries for wily programmers who wish to develop X applications. Widget sets like Athena, Open Look, and Xaw3D are supported. All of the standard fonts, bitmaps, manual pages, and documentation are included. PEX (a programming interface for 3-dimensional graphics) is also supported.

Many X application programmers use the proprietary Motif widget set for development. Several vendors sell single and multiple user licenses for binary versions of Motif. Because Motif itself is relatively expensive, not many Linux users own it. However, binaries statically linked with Motif routines can be freely distributed. If you write a program using Motif, you may provide a binary so users without the Motif libraries can use the program.

A major caveat to using the X Window System is its hardware requirements. A 80386-based CPU with 4 megabytes of RAM is capable of running X, but 16 megabytes or more of physical RAM is needed for comfortable use. A faster processor is nice to have as well, but having enough physical RAM is much more important. In addition, to achieve really slick video performance, we recommend getting an accelerated video card, like a VESA Local Bus (VLB) S3 chipset card. Performance ratings in excess of 300,000 xstones have been achieved with Linux and XFree86. Using adequate hardware, you'll find that running X and Linux is as fast, or faster, than running X on other UNIX workstations.

In Chapter **??** we discuss how to install and use X on your system.

1.4.4 Introduction to Networking.

Would you like to communicate with the world? Linux supports two primary UNIX networking protocols: TCP/IP and UUCP. TCP/IP (Transmission Control Protocol/Internet Protocol) is the networking paradigm which allows systems all over the world to communicate on a single network, the **Internet.** With Linux, TCP/IP, and a connection to the Internet, you can communicate with users and machines via electronic mail, Usenet news, and FTP file transfer.

Most TCP/IP networks use Ethernet as the physical network transport. Linux supports

many popular Ethernet cards and interfaces for personal computers, including pocket and PCMCIA Ethernet adaptors.

However, because not everyone has an Ethernet connection at home, Linux also supports **SLIP** (Serial Line Internet Protocol) and **PPP** (Point-to-Point Protocol), which provide Internet access via modem. Many businesses and universities provide SLIP and PPP servers. In fact, if your Linux system has an Ethernet connection to the Internet and a modem, your system can become a SLIP or PPP server for other hosts.

NFS (Network File System) lets your system seamlessly share file systems with other machines on the network. FTP (File Transfer Protocol) lets you transfer files with other machines. `sendmail` sends and receives electronic mail via the SMTP protocol; C-News and INN are NNTP based new systems; and `telnet`, `rlogin`, and `rsh` let you log in and execute commands on other machines on the network. `finger` lets you get information about other Internet users.

Linux also supports Microsoft Windows connectivity via Samba[2] , and Macintosh connectivity with AppleTalk and LocalTalk. Support for Novell's IPX protocol is also included.

The full range of mail and news readers is available for Linux, including `elm`, `pine`, `rn`, `nn`, and `tin`. Whatever your preference, you can configure a Linux system to send and receive electronic mail and news from all over the world.

The system provides a standard UNIX socket programming interface. Virtually any program that uses TCP/IP can be ported to Linux. The Linux X server also supports TCP/IP, and applications running on other systems may use the display of your local system.

In Chapter **??**, we discuss the installation of TCP/IP software, including SLIP and PPP.

UUCP (UNIX-to-UNIX Copy) is an older mechanism to transfer files, electronic mail, and electronic news between UNIX machines. Historically, UUCP machines are connected over telephone lines via modem, but UUCP is able to transfer data over a TCP/IP network as well. If you do not have access to a TCP/IP network or a SLIP or PPP server, you can configure your system to send and receive files and electronic mail using UUCP. See Chapter **??** for more information.

1.4.5 Telecommunications and BBS software.

If you have a modem, you'll be able to communicate with other machines via telecommunications packages available for Linux. Many people use telecommunications software to access bulletin board systems (BBS's) as well as commercial, online services like

[2] See *Samba: Integrating UNIX and Windows*, Copyright 1998 Specialized Systems Consultants.

Prodigy, CompuServe, and America Online. People use modems to connect to UNIX systems at work or school. Modems can send and receive faxes.

A popular communications package for Linux is `seyon,` which provides a customizable, ergonomic interface undex X and has built-in support for the Kermit and ZModem file transfer protocols. Other telecommunications programs include C-Kermit, `pcomm`, and `minicom`. These are similar to communications programs found on other operating systems, and are quite easy to use.

If you do not have access to a SLIP or PPP server (see the previous section), you can use `term` to multiplex your serial line. The `term` program allows you to open more than one login session over a modem connection. It lets you redirect X client connections to your local X server via a serial line. Another software package, KA9Q, implements a similar, SLIP-like interface.

Operating a Bulletin Board System (BBS) is a favorite hobby and means of income for many people. Linux supports a wide range of BBS software, most of which is more powerful than that available for other operating systems. With a phone line, modem, and Linux, you can turn your system into a BBS and provide dial-in access for users worldwide. BBS software for Linux includes XBBS and UniBoard BBS packages.

Most BBS software locks the user into a menu based system where only certain functions and applications are available. An alternative to BBS access is full UNIX access, which lets users dial into your system and log in normally. This requires a fair amount of maintenance by the system administrator, but providing public UNIX access is not difficult. In addition to TCP/IP networking, you can make electronic mail and news access available on your system.

If you do not have access to a TCP/IP network or UUCP feed, Linux lets you communicate with BBS networks like FidoNet, which let you exchange electronic news and mail over a telephone line. You can find more information on telecommunications and BBS software under Linux in Chapter **??**.

1.4.6 World Wide Web.

It is worth noting that Linux includes web server software as well as web browsers. The most common server is Apache. Thousands of Linux systems run Apache on the Internet today, including the Linux Resources site, `www.linuxresources.com`.

Linux distributions include different web browsers, and other browsers can be downloaded from the Internet. Available browsers include Lynx, Mosaic, Netscape, Arena, and Amaya.

Linux provides complete support for Java and CGI applets, and Perl is a standard tool in the Linux programming environment.

1.4.7 Interfacing and MS-DOS.

Various utilities exist to interface with MS-DOS. The most well-known application is the Linux MS-DOS Emulator, which lets you run MS-DOS applications directly from Linux. Although Linux and MS-DOS are completely different operating systems, the 80386 protected-mode environment allows MS-DOS applications to behave as if they were running in their native 8086 environment.

The MS-DOS emulator is still under development, but many popular applications run under it. Understandably, MS-DOS applications that use bizarre or esoteric features of the system may never be supported, because of the limitations inherent in any emulator. For example, you shouldn't expect to run programs that use 80386 protected-mode features, like Microsoft Windows (in 386 enhanced mode, that is).

Standard MS-DOS commands and utilities like `PKZIP.EXE` work under the emulators, as do 4DOS, a `COMMAND.COM` replacement, FoxPro 2.0, Harvard Graphics, MathCad, Stacker 3.1, Turbo Assembler, Turbo C/C++, Turbo Pascal, Microsoft Windows 3.0 (in real mode), and WordPerfect 5.1.

The MS-DOS Emulator is meant mostly as an ad-hoc solution for those who need MS-DOS for only a few applications and use Linux for everything else. It's not meant to be a complete implementation of MS-DOS. Of course, if the Emulator doesn't satisfy your needs, you can always run MS-DOS as well as Linux on the same system. Using the LILO boot loader, you can specify at boot time which operating system to start. Linux can also coexist with other operating systems, like OS/2.

Linux provides a seamless interface to transfer files between Linux and MS-DOS. You can mount a MS-DOS partition or floppy under Linux, and directly access MS-DOS files as you would any file.

Currently under development is **WINE**—a Microsoft Windows emulator for the X Window System under Linux. Once WINE is complete, users will be able to run MS-Windows applications directly from Linux. This is similar to the commercial WABI Windows emulator from Sun Microsystems, which is also available for Linux.

In Chapter **??**, we talk about the MS-DOS tools available for Linux.

1.4.8 Other applications.

A host of miscellaneous programs and utilities exist for Linux, as one would expect of such a hodgepodge operating system. Linux's primary focus is UNIX personal computing, but this is not the only field where it excels. The selection of business and scientific software is expanding, and commercial software vendors have begun to contribute to the growing pool Linux applications.

Several relational databases are available for Linux, including Postgres, Ingres, and Mbase. These are full-featured, professional, client/server database applications, similar to those found on other UNIX platforms. Many commercial database systems are available as well.

Scientific computing applications include FELT (finite element analysis); `gnuplot` (data plotting and analysis); Octave (a symbolic mathematics package similar to MATLAB); `xspread` (a spreadsheet calculator); `xfractint` (an X-based port of the popular Fractint fractal generator); and `xlispstat` (statistics). Other applications include SPICE (circuit design and analysis) and Khoros (image and digital signal processing and visualization). Commercial packages like Maple and MathLab are available.

Many more applications have been ported to Linux. If you absolutely cannot find what you need, you can attempt to port the application from another platform to Linux yourself. Whatever your field, porting standard UNIX applications to Linux is straightforward. Linux's complete UNIX programming environment is sufficient to serve as the base for any scientific application.

Linux also has its share of games. These include classic text based dungeon games like Nethack and Moria; **MUDs** (multi-user dungeons, which allow many users to interact in a text-based adventure) like DikuMUD and TinyMUD; and a slew of X games like `xtetris`, `netrek`, and `xboard`, the X11 version of `gnuchess`. The popular shoot-em-up, arcade-style game, Doom, has also been ported to Linux.

For audiophiles, Linux supports various sound cards and related software, like CD-player, which makes a CD-ROM drive into an audio CD player, MIDI sequencers and editors, which let you compose music for playback through a synthesizer or other MIDI controlled instrument, and sound editors for digitized sounds.

Can't find the application you're looking for? The Linux Software Map, described in Appendix A, lists software packages which have been written or ported to Linux. Another way to find Linux applications is to look at the `INDEX` files found on Linux FTP sites, if you have Internet access.

Most freely-distributable, UNIX based software will compile on Linux with little difficulty. If all else fails, you can write the application yourself. If you're looking for a

commercial application, there may be a free "clone" available. Or, you can encourage the software company to consider releasing a binary version for Linux. Several individuals have contacted software companies and asked them to port their applications to Linux, with various degrees of success.

1.5 Copyright issues.

Linux is covered by what is known as the GNU *General Public License*, or **GPL**. The GPL was developed for the GNU project by the Free Software Foundation and specifies several provisions for the distribution and modification of free software. *Free*, in this sense, refers to distribution, not cost. The GPL has always been subject to misinterpretation. We hope that this summary will help you understand the extent and goals of the GPL and its effect on Linux. A complete copy of the GPL is printed in Appendix C.

Originally, Linus Torvalds released Linux under a license more restrictive than the GPL, which allowed the software to be freely distributed and modified, but prevented any money from changing hands for its distribution and use. On the other hand, the GPL allows people to sell and profit from free software, but does not allow them to restrict another's right to distribute the software in any way.

First, it should be explained that free software that is covered by the GPL is not in the public domain. Public domain software by definition is not copyrighted and is literally owned by the public. Software covered by the GPL, on the other hand, is copyrighted by the author. The software is protected by standard international copyright laws, and the author is legally defined. The GPL provides for software which may be freely distributed but is not in the public domain.

GPL-licensed software is also not shareware. Generally, shareware is owned and copyrighted by an author who requires users to send in money for its use. Software covered by the GPL may be distributed and used free of charge.

The GPL also lets people take, modify, and distribute their own versions of the software. However, any derived works of GPL software must also be covered by the GPL. In other words, a company may not take Linux, modify it, and sell it under a restrictive license. If the software is derived from Linux, that software must be covered under the GPL also.

The GPL allows free software to be distributed and used free of charge. It also lets a person or organization distribute GPL software for a fee, and even make a profit from its sale and distribution. However, a distributor of GPL software cannot take those rights away from a purchaser. If you purchase GPL software from a third-party source, you may distribute the software for free, and sell it yourself as well.

This may sound like a contradiction. Why sell software when the GPL allows you to get it for free? Let's say that a company decided to bundle a large amount of free software on a CD-ROM and distribute it. That company would need to charge for the overhead of producing and distributing the CD-ROM, and may even decide to profit from the sales of the software. This is allowed by the GPL.

Organizations that sell free software must follow certain restrictions set forth in the GPL. They cannot restrict the rights of users who purchase the software. If you buy a CD-ROM that contains GPL software, you can copy and distribute the CD-ROM free of charge, or resell it yourself. Distributors must make obvious to users that the software is covered by the GPL. Distributors must also provide, free of charge, the complete source code to the software distributed. This permits anyone who purchases GPL software to make modifications to that software.

Allowing a company to distribute and sell free software is a good thing. Not everyone has access to the Internet and the ability to download software for free. Many organizations sell Linux on diskette, tape, or CD-ROM via mail order, and profit from the sales. Linux developers may never see any of this profit; that is the understanding reached between the developer and the distributor when software is licensed by the GPL. In other words, Linus Torvalds knew that companies may wish to sell Linux, and that he might not see a penny of the profits.

In the free software world, the important issue is not money. The goal of free software is always to develop and distribute fantastic software and allow anyone to obtain and use it. In the next section, we'll discuss how this applies to the development of Linux.

1.6 The design and philosophy of Linux.

New users often have a few misconceptions and false expectations about Linux. It is important to understand the philosophy and design of Linux in order to use it effectively. We'll start by describing how Linux is *not* designed.

In commercial UNIX development houses, the entire system is developed under a rigorous quality assurance policy that utilizes source and revision control systems, documentation, and procedures to report and resolve bugs. Developers may not add features or change key sections of code on a whim. They must validate the change as a response to a bug report and subsequently "check in" all changes to the source control system, so that the changes may be reversed if necessary. Each developer is assigned one or more parts of the system code, and only that developer can alter those sections of the code while it is "checked out" (that is, while the code is under his or her control).

Organizationally, a quality assurance department runs rigorous tests on each new version of the operating system and reports any bugs. The developers fix these bugs as reported. A complex system of statistical analysis is used to ensure that a certain percentage of bugs are fixed before the next release, and that the operating system as a whole passes certain release criteria.

The software company, quite reasonably, must have quantitative proof that the next revision of the operating system is ready to be shipped; hence, the gathering and analysis of statistics about the performance of the operating system. It is a big job to develop a commercial UNIX system, often large enough to employ hundreds, if not thousands, of programmers, testers, documenters, and administrative personnel. Of course, no two commercial UNIX vendors are alike, but that is the general picture.

The Linux model of software development discards the entire concept of organized development, source code control systems, structured bug reporting, and statistical quality control. Linux is, and likely always will be, a hacker's operating system. (By *hacker*, I mean a feverishly dedicated programmer who enjoys exploiting computers and does interesting things with them. This is the original definition of the term, in contrast to the connotation of *hacker* as a computer wrongdoer, or outlaw.)

There is no single organization responsible for developing Linux. Anyone with enough know-how has the opportunity to help develop and debug the kernel, port new software, write documentation, and help new users. For the most part, the Linux community communicates via mailing lists and Usenet newsgroups. Several conventions have sprung up around the development effort. Anyone who wishes to have their code included in the "official" kernel, mails it to Linus Torvalds. He will test and include the code in the kernel as long as it doesn't break things or go against the overall design of the system.

The system itself is designed using an open-ended, feature-minded approach. The number of new features and critical changes to the system has recently diminished, and the general rule is that a new version of the kernel will be released every few weeks. Of course, this is a rough figure. New release criteria include the number of bugs to be fixed, feedback from users testing pre-release versions of the code, and the amount of sleep Linus Torvalds has had this week.

Suffice it to say that not every bug is fixed, nor is every problem ironed out between releases. As long as the revision appears to be free of critical or recurring bugs, it is considered to be stable, and the new version is released. The thrust behind Linux development is not to release perfect, bug-free code: it is to develop a free UNIX implementation. Linux is for the developers, more than anyone else.

Anyone who has a new feature or software application generally makes it available in an

alpha version—that is, a test version, for those brave users who want to hash out problems in the initial code. Because the Linux community is largely based on the Internet, alpha software is usually uploaded to one or more Linux FTP sites (see Appendix B), and a message is posted to one of the Linux Usenet newsgroups about how to obtain and test the code. Users who download and test alpha software can then mail results, bug fixes, and questions to the author.

After the initial bugs have been fixed, the code enters a **beta test** stage, in which it is usually considered stable but not complete. It works, but not all of the features may be present. The software may also go directly to a final stage, in which the software is considered complete and usable.

Keep in mind that these are only conventions—not rules. Some developers may feel so confident of their software that they decide it isn't necessary to release alpha or test versions. It is always up to the developer to make these decisions.

You might be amazed at how such an unstructured system of volunteers who program and debug a complete UNIX system gets anything done at all. As it turns out, this is one of the most efficient and motivated development efforts ever employed. The entire Linux kernel is written *from scratch*, without code from proprietary sources. It takes a huge amount of work to port all the free software under the sun to Linux. Libraries are written and ported, file systems are developed, and hardware drivers are written for many popular devices—all due to the work of volunteers.

Linux software is generally released as a **distribution**, a set of prepackaged software which comprises an entire system. It would be difficult for most users to build a complete system from the ground up, starting with the kernel, adding utilities, and installing all of the necessary software by hand. Instead, many software distributions are available which include everything necessary to install and run a complete system. There is no single, standard distribution—there are many, and each has its own advantages and disadvantages. We describe installation of the various Linux distributions starting on page 53.

1.7 Differences between Linux and other operating systems.

It is important to understand the differences between Linux and other operating systems, like MS-DOS, OS/2, and the other implementations of UNIX for personal computers. First of all, Linux coexists happily with other operating systems on the same machine: you can run MS-DOS and OS/2 along with Linux on the same system without problems. There

are even ways to interact between various operating systems, as we'll see.

Why use Linux? Why use Linux, instead of a well known, well tested, and well documented commercial operating system? We could give you a thousand reasons. One of the most important, however, is that Linux is an excellent choice for personal UNIX computing. If you're a UNIX software developer, why use MS-DOS at home? Linux allows you to develop and test UNIX software on your PC, including database and X Window System applications. If you're a student, chances are that your university computing systems run UNIX. You can run your own UNIX system and tailor it to your needs. Installing and running Linux is also an excellent way to learn UNIX if you don't have access to other UNIX machines.

But let's not lose sight. Linux isn't only for personal UNIX users. It is robust and complete enough to handle large tasks, as well as distributed computing needs. Many businesses—especially small ones—have moved their systems to Linux in lieu of other UNIX based, workstation environments. Universities have found that Linux is perfect for teaching courses in operating systems design. Large, commercial software vendors have started to realize the opportunities which a free operating system can provide.

Linux vs. MS-DOS. It's not uncommon to run both Linux and MS-DOS on the same system. Many Linux users rely on MS-DOS for applications like word processing. Linux provides its own analogs for these applications, but you might have a good reason to run MS-DOS as well as Linux. If your dissertation is written using WordPerfect for MS-DOS, you may not be able to convert it easily to TEX or some other format. Many commercial applications for MS-DOS aren't available for Linux yet, but there's no reason that you can't use both.

MS-DOS does not fully utilize the functionality of 80386 and 80486 processors. On the other hand, Linux runs completely in the processor's protected mode, and utilizes all of its features. You can directly access all of your available memory (and beyond, with virtual RAM). Linux provides a complete UNIX interface which is not available under MS-DOS. You can easily develop and port UNIX applications to Linux, but under MS-DOS you are limited to a subset of UNIX functionality.

Linux and MS-DOS are different entities. MS-DOS is inexpensive compared to other commercial operating systems and has a strong foothold in the personal computer world. No other operating system for the personal computer has reached the level of popularity of MS-DOS, because justifying spending $1,000 for other operating systems alone is unrealistic for many users. Linux, however, is free, and you may finally have the chance to decide

for yourself.

You can judge Linux vs. MS-DOS based on your expectations and needs. Linux is not for everybody. If you always wanted to run a complete UNIX system at home, without the high cost of other UNIX implementations for personal computers, Linux may be what you're looking for.

Linux vs. The Other Guys. A number of other advanced operating systems have become popular in the PC world. Specifically, IBM's OS/2 and Microsoft Windows have become popular for users upgrading from MS-DOS.

Both OS/2 and Windows NT are full featured multitasking operating systems, like Linux. OS/2, Windows NT, and Linux support roughly the same user interface, networking, and security features. However, the real difference between Linux and The Other Guys is the fact that Linux is a version of UNIX, and benefits from contributions of the UNIX community at large.

What makes UNIX so important? Not only is it the most popular operating system for multiuser machines, it is a foundation of the free software world. Much of the free software available on the Internet is written specifically for UNIX systems.

There are many implementations of UNIX from many vendors. No single organization is responsible for its distribution. There is a large push in the UNIX community for standardization in the form of open systems, but no single group controls this design. Any vendor (or, as it turns out, any hacker) may develop a standard implementation of UNIX.

OS/2 and Microsoft operating systems, on the other hand, are proprietary. The interface and design are controlled by a single corporation, which develops the operating system code. In one sense, this kind of organization is beneficial because it sets strict standards for programming and user interface design, unlike those found even in the open systems community.

Several organizations have attempted the difficult task of standardizing the UNIX programming interface. Linux, in particular, is mostly compliant with the POSIX.1 standard. As time goes by, it is expected that the Linux system will adhere to other standards, but standardization is not the primary goal of Linux development.

Linux vs. other implementations of UNIX. Several other implementations of UNIX exist for 80386 or better personal computers. The 80386 architecture lends itself to UNIX, and vendors have taken advantage of this.

Oher implementations of UNIX for the personal computer are similar to Linux. Almost all commercial versions of UNIX support roughly the same software, programming

environment, and networking features. However, there are differences between Linux and commercial versions of UNIX.

Linux supports a different range of hardware than commercial implementations. In general, Linux supports most well-known hardware devices, but support is still limited to hardware which the developers own. Commercial UNIX vendors tend to support more hardware at the outset, but the list of hardware devices which Linux supports is expanding continuously. We'll cover the hardware requirements for Linux in Section 1.8.

Many users report that Linux is at least as stable as commercial UNIX systems. Linux is still under development, but the two-pronged release philosophy has made stable versions available without impeding development.

The most important factor for many users is price. Linux software is free if you can download it from the Internet or another computer network. If you do not have Internet access, you can still purchase Linux inexpensively via mail order on diskette, tape, or CD-ROM.

Of course, you may copy Linux from a friend who already has the software, or share the purchase cost with someone else. If you plan to install Linux on a large number of machines, you need only purchase a single copy of the software—Linux is not distributed with a "single machine" license.

The value of commercial UNIX implementations should not be demeaned. In addition to the price of the software itself, one often pays for documentation, support, and quality assurance. These are very important factors for large institutions, but personal computer users may not require these benefits. In any case, many businesses and universities have found that running Linux in a lab of inexpensive personal computers is preferable to running a commercial version of UNIX in a lab of workstations. Linux can provide workstation functionality on a personal computer at a fraction of the cost.

Linux systems have travelled the high seas of the North Pacific, and manage telecommunications and data analysis for an oceanographic research vessel. Linux systems are used at research stations in Antarctica. Several hospitals maintain patient records on Linux systems.

Other free or inexpensive implementations of UNIX are available for the 80386 and 80486. One of the best known is 386BSD, an implementation of BSD UNIX for the 80386. The 386BSD package is comparable to Linux in many ways, but which one is better depends on your needs and expectations. The only strong distinction we can make is that Linux is developed openly, and any volunteer can aid in the development process, while 386BSD is developed by a closed team of programmers. Because of this, serious philosophical and design differences exist between the two projects. The goal of Linux is to

develop a complete UNIX system from scratch (and have a lot of fun in the process), and the goal of 386BSD is in part to modify the existing BSD code for use on the 80386.

NetBSD is another port of the BSD NET/2 distribution to several machines, including the 80386. NetBSD has a slightly more open development structure, and is comparable to 386BSD in many respects.

Another project of note is HURD, an effort by the Free Software Foundation to develop and distribute a free version of UNIX for many platforms. Contact the Free Software Foundation (the address is given in Appendix C) for more information about this project. At the time of this writing, HURD is still under development.

Other inexpensive versions of UNIX exist as well, like Minix, an academic but useful UNIX clone upon which early development of Linux was based. Some of these implementations are mostly of academic interest, while others are full fledged systems.

1.8 Hardware requirements.

You must be convinced by now of how wonderful Linux is, and of all the great things it can do for you. However, before you rush out and install Linux, you need to be aware of its hardware requirements and limitations.

Keep in mind that Linux is developed by users. This means, for the most part, that the hardware supported by Linux is that which the users and developers have access to. As it turns out, most popular hardware and peripherals for personal computers are supported. Linux supports more hardware than some commercial implementations of UNIX. However, some obscure devices aren't supported yet.

Another drawback of hardware support under Linux is that many companies keep their hardware interfaces proprietary. Volunteer Linux developers can't write drivers for the devices because the manufacturer does not make the technical specifications public. Even if Linux developers could develop drivers for proprietary devices, they would be owned by the company which owns the device interface, which violates the GPL. Manufacturers that maintain proprietary interfaces write their own drivers for operating systems like MS-DOS and Microsoft Windows. Users and third-party developers never need to know the details of the interface.

In some cases, Linux programmers have attempted to write hackish device drivers based on assumptions about the interface. In other cases, developers work with the manufacturer and try to obtain information about the device interface, with varying degrees of success.

In the following sections, we attempt to summarize the hardware requirements for Linux. The Linux Hardware HOWTO (see Section 1.9) contains a more complete listing

of hardware supported by Linux.

◇ **Disclaimer:** Much hardware support for Linux is in the development stage. Some distributions may or may not support experimental features. This section lists hardware which has been supported for some time and is known to be stable. When in doubt, consult the documentation of your Linux distribution. See Section 2.2 for more information about Linux distributions.

Linux is available for many platforms in addition to Intel 80x86 systems. These include Macintosh, Amiga, Sun SparcStation, and Digital Equipment Corporation Alpha based systems. In this book, however, we focus on garden-variety Intel 80386, 80486, and Pentium processors, and clones by manufacturers like AMD, Cyrix, and IBM.

Motherboard and CPU requirements. Linux currently supports systems with the Intel 80386, 80486, or Pentium CPU, including all variations like the 80386SX, 80486SX, 80486DX, and 80486DX2. Non-Intel clones work with Linux as well. Linux has also been ported to the DEC Alpha and the Apple PowerMac.

If you have an 80386 or 80486SX, you may also wish to use a math coprocessor, although one isn't required. The Linux kernel can perform FPU emulation if the machine doesn't have a coprocessor. All standard FPU couplings are supported, including IIT, Cyrix FasMath, and Intel.

Most common PC motherboards are based on the PCI bus but also offer ISA slots. This configuration is supported by Linux, as are EISA and VESA-bus systems. IBM's MicroChannel (MCA) bus, found on most IBM PS/2 systems, is significantly different, and support has been recently added.

Memory requirements. Linux requires very little memory, compared to other advanced operating systems. You should have 4 megabytes of RAM at the very least, and 16 megabytes is strongly recommended. The more memory you have, the faster the system will run. Some distributions require more RAM for installation.

Linux supports the full 32-bit address range of the processor. In other words, it uses all of your RAM automatically.

Linux will run with only 4 megabytes of RAM, including bells and whistles like the X Window System and `emacs`. However, having more memory is almost as important as having a faster processor. For general use, 16 megabytes is enough, and 32 megabytes, or more, may be needed for systems with a heavy user load.

Most Linux users allocate a portion of their hard drive as swap space, which is used as **virtual RAM**. Even if your machine has more than 16 megabytes of physical RAM, you

may wish to use swap space. It is no replacement for physical RAM, but it can let your system run larger applications by swapping inactive portions of code to disk. The amount of swap space that you should allocate depends on several factors; we'll come back to this question in Chapter 2.

Hard drive controller requirements. It is possible to run Linux from a floppy diskette, or, for some distributions, a live file system on CD-ROM, but for good performance you need hard disk space. Linux can co-exist with other operating systems—it only needs one or more disk partitions.

Linux supports all IDE and EIDE controllers as well as older MFM and RLL controllers. Most, but not all, ESDI controllers are supported. The general rule for non-SCSI hard drive and floppy controllers is that if you can access the drive from MS-DOS or another operating system, you should be able to access it from Linux.

Linux also supports a number of popular SCSI drive controllers. This includes most Adaptec and Buslogic cards as well as cards based on the NCR chip sets.

Hard drive space requirements. Of course, to install Linux, you need to have some amount of free space on your hard drive. Linux will support more than one hard drive on the same machine; you can allocate space for Linux across multiple drives if necessary.

How much hard drive space depends on your needs and the software you're installing. Linux is relatively small, as UNIX implementations go. You could run a system in 20 megabytes of disk space. However, for expansion and larger packages like X, you need more space. If you plan to let more than one person use the machine, you need to allocate storage for their files. Realistic space requirements range from 200 megabytes to one gigabyte or more.

Also, you will likely want to allocate disk space as virtual RAM. We will discuss installing and using swap space in Chapter 2.

Each Linux distribution comes with literature to help you gauge the precise amount of storage required for your software configuration. Look at the information which comes with your distribution or the appropriate installation section in Chapter 2.

Monitor and video adaptor requirements. Linux supports standard Hercules, CGA, EGA, VGA, IBM monochrome, Super VGA, and many accelerated video cards, and monitors for the default, text-based interface. In general, if the video card and monitor work under an operating system like MS-DOS, the combination should work fine under Linux. However, original IBM CGA cards suffer from "snow" under Linux, which is not pleasant

to view.

Graphical environments like X have video hardware requirements of their own. Rather than list them here, we relegate that discussion to Section 5.1. Popular video cards are supported and new card support is added regularly.

Miscellaneous hardware. You may also have devices like a CD-ROM drive, mouse, or sound card, and may be interested in whether or not this hardware is supported by Linux.

Mice and other pointing devices. Typically, a mouse is used only in graphical environments like X. However, several Linux applications that are not associated with a graphical environment also use mice.

Linux supports standard serial mice like Logitech, MM series, Mouseman, Microsoft (2-button), and Mouse Systems (3-button). Linux also supports Microsoft, Logitech, and ATIXL bus mice, and the PS/2 mouse interface.

Pointing devices that emulate mice, like trackballs and touchpads, should work also.

CD-ROM drives. Many common CD-ROM drives attach to standard IDE controllers. Another common interface for CD-ROM is SCSI. SCSI support includes multiple logical units per device so you can use CD-ROM "jukeboxes." Additionally, a few proprietary interfaces, like the NEC CDR-74, Sony CDU-541 and CDU-31a, Texel DM-3024, and Mitsumi are supported.

Linux supports the standard ISO 9660 file system for CD-ROMs, and the High Sierra file system extensions.

Tape drives. Any SCSI tape drive, including quarter inch, DAT, and 8MM are supported, if the SCSI controller is supported. Devices that connect to the floppy controller like floppy tape drives are supported as well, as are some other interfaces, like QIC-02.

Printers. Linux supports the complete range of parallel printers. If MS-DOS or some other operating system can access your printer from the parallel port, Linux should be able to access it, too. Linux printer software includes the UNIX standard `lp` and `lpr` software. This software allows you to print remotely via a network, if you have one. Linux also includes software that allows most printers to handle PostScript files.

Modems. As with printer support, Linux supports the full range of serial modems, both internal and external. A great deal of telecommunications software is available for

Linux, including Kermit, `pcomm`, `minicom`, and `seyon`. If your modem is accessible from another operating system on the same machine, you should be able to access it from Linux with no difficulty.

Ethernet cards. Many popular Ethernet cards and LAN adaptors are supported by Linux. Linux also supports some FDDI, frame relay, and token ring cards, and all Arcnet cards. A list of supported network cards is included in the kernel source of your distribution.

1.9 Sources of Linux information.

Many other sources of information about Linux are available. In particular, a number of books about UNIX in general will be of use, especially for readers unfamiliar with UNIX. We suggest that you peruse one of these books before attempting to brave the jungles of Linux.

Information is also available online in electronic form. You must have access to an online network like the Internet, Usenet, or Fidonet to access the information. A good place to start is `www.linuxresources.com` (see Appendix A). If you do not, you might be able to find someone who is kind enough to give you hard copies of the documents.

1.9.1 Online documents.

Many Linux documents are available via anonymous FTP from Internet archive sites around the world and networks like Fidonet and CompuServe. Linux CD-ROM distributions also contain the documents mentioned here. If you are can send mail to Internet sites, you may be able to retrieve these files using one of the FTP e-mail servers that mail you the documents or files from the FTP sites. See Appendix B for more information on using FTP e-mail servers.

A list of well-known Linux archive sites is given in Appendix B. To reduce network traffic, you should use a FTP site that is geographically close to you.

Appendix A contains a partial list of the Linux documents available via anonymous FTP. The filenames vary depending on the site. Most sites keep Linux-related documents in the `docs` subdirectory of their Linux archive. For example, the FTP site `sunsite.unc.edu`, keeps Linux files in `/pub/Linux`, with Linux-related documentation in `/pub/Linux/docs`.

Examples of available online documents are *Linux Frequently Asked Questions with Answers*, a collection of frequently asked questions about Linux; Linux HOWTO docu-

ments, which describe specific aspects of the system, like the Installation HOWTO, Printing HOWTO, and Ethernet HOWTO; and the *Linux META-FAQ*, which is a list of information sources on the Internet.

Many of these documents are also regularly posted to one or more Linux-related Usenet newsgroups; see Section 1.9.4 below.

1.9.2 Linux on the World Wide Web.

The Linux Documentation Project Home Page is on the World Wide Web at `http://sunsite.unc.edu/LDP` This web page lists many HOWTOs and other documents in HTML format, as well as pointers to other sites of interest to Linux users, like `ssc.com`, home of the *Linux Journal*, a monthly magazine. You can find their home page at `http://www.ssc.com/`.

1.9.3 Books and other published works.

The books of the Linux Documentation Project are the result of an effort carried out over the Internet to write and distribute a bona fide set of manuals for Linux, analogs of the documentation which comes with commercial UNIX versions and covers installation, operation, programming, networking, and kernel development.

Linux Documentation Project manuals are available via anonymous FTP and by mail order. Appendix A lists the manuals available and describes how to obtain them.

Many large publishers, including MIS:Press, Digital Press, O'Reilly & Associates, and SAMS have jumped onto the Linux bandwagon. Check with computer bookstores or SSC's web page at `http://www.ssc.com/`, or the book reviews in *Linux Journal*, sometimes made available on their site, `http://www.linuxjournal.com`

A large number of books about UNIX in general are applicable to Linux. In its use and programming interface, Linux does not differ greatly from other implementations of UNIX. Almost everything you would like to know about using and programming Linux can be found in general UNIX texts. In fact, this book is meant to supplement the library of UNIX books currently available. Here, we present the most important Linux-specific details and hope that you will look to other sources for in-depth information.

Armed with good books about UNIX as well as this book, you should be able to tackle just about anything. Appendix A lists several UNIX books which are recommended highly for UNIX newcomers and wizards.

The *Linux Journal* magazine is distributed worldwide, and is an excellent way to keep in touch with the goings-on of the Linux community, especially if you do not have access

to Usenet news (see below). See Appendix A for information on subscribing to the *Linux Journal*.

1.9.4 Usenet newsgroups.

Usenet is a worldwide electronic news and discussion forum with a diverse selection of **newsgroups,** which are discussion areas devoted to specific topics. Much discussion about Linux development occurrs over the Internet and Usenet. Not surprisingly, a number of Usenet newsgroups are dedicated to Linux.

The original Linux newsgroup, `alt.os.linux`, was created to move some of the discussion about Linux from `comp.os.minix` and various mailing lists. Soon, the traffic on `alt.os.linux` grew large enough that a newsgroup in the `comp` hierarchy was warranted. A vote was taken in February, 1992, and `comp.os.linux` was created.

`comp.os.linux` quickly became one of the most popular (and loudest) of the Usenet groups, more popular than any other group in the `comp.os` hierarchy. In December, 1992, a vote was taken to split the newsgroup to reduce traffic; only `comp.os.linux.announce` passed this vote. In July, 1993, the group was finally split into a new hierarchy. Almost 2,000 people voted in the `comp.os.linux` reorganization, making it one of the largest Usenet Calls For Votes ever.

If you do not have Usenet, there are mail-to-news gateways available for many (if not all) of the newsgroups below.

`comp.os.linux.advocacy`
: A newsgroup for discussion of the benefits of Linux compared to other operating systems.

`comp.os.linux.alpha`
: The comp.os.linux.alpha newsgroup should be used for all discussions relating to buying, installing, running, maintaining, and developing Linux on Digital Alpha processor based systems.

`comp.os.linux.announce`
: A moderated newsgroup for announcements about Linux, including bug reports and important patches to software. If you read any Linux newsgroup at all, read this one. Often, the important postings in this group are not crossposted. This group also contains many periodic postings about Linux, including the online documents described in the last section and listed in Appendix A.

Postings to these newsgroups must be approved by the moderators, Matt Welsh and Lars Wirzenius. If you wish to submit an article, you simply post the article as you normally would; the news software will forward the article to the moderators for approval. However, if your news system is not set up correctly, you may need to mail the article directly to `linux-announce@tc.cornell.edu`.

`comp.os.linux.answers`

For posting Linux FAQs, How-To's, READMEs and other documents that answer questions about Linux. This will help keep the traffic down in other c.o.l.* groups and will leave comp.os.linux.announce for true announcements.

`comp.os.linux.development.apps`

An unmoderated newsgroup for questions and discussion regarding the writing of applications for Linux and the porting of applications to Linux.

`comp.os.linux.development.system`

An unmoderated newsgroup for discussions about the development of the Linux system related to the kernel, device drivers, and loadable modules.

`comp.os.linux.hardware`

This newsgroup is for questions and discussion specific to a particular piece of hardware, e.g., "can this system run Linux?", "how do I use this disk drive with Linux?", etc.

`comp.os.linux.m68k`

This is to further interest in and development of the port of Linux to Motorola 680x0 architecture.

`comp.os.linux.misc`

All discussion which doesn't quite fit into the other available Linux groups. Any nontechnical or metadiscourse about Linux should occur in `comp.os.linux.misc`.

`comp.os.linux.networking`

Discussion relating to networking and communications including Ethernet boards, SLIP, and PPP.

`comp.os.linux.setup`
: Questions and discussion relating to Linux installation and system administration.

`comp.os.linux.x`
: Discussion of X Window System features unique to Linux, including servers, clients, fonts, and libraries.

This list is by no means complete. New groups are created when a need for a subdivision of discussion is advisable, and there are linux groups in other hierarchies as well.

1.9.5 Internet mailing lists.

If you have access to Internet electronic mail, you can participate in several mailing lists, even if you do not have Usenet access. If you are not directly on the Internet, you can join one of these mailing lists if you can exchange electronic mail with the Internet (for example, through UUCP, Fidonet, CompuServe, or other networks which exchange Internet mail).

For more information about the Linux mailing lists, send e-mail to

`majordomo@vger.rutgers.edu`

Include a line with the word `help` in the body of the message, and a message will be returned to you which describes how to subscribe and unsubscribe to various mailing lists. The word `lists` on a line by itself will retrieve the names of mailing lists which are accessible through the `majordomo.vger.rutgers.edu` server.

There are several special-purpose mailing lists for Linux as well. The best way to find out about these is to watch the Linux Usenet newsgroups for announcements, as well as to read the list of publicly-available mailing lists, which is posted to the Usenet `news.answers` group.

1.10 Getting Help with Linux.

You will undoubtedly need assistance during your adventures in the Linux world. Even UNIX wizards are occasionally stumped by some quirk or feature of Linux. It's important to know how, where, and when to find help.

The primary means of obtaining help is through Internet mailing lists and newsgroups as discussed in Section 1.9. If you don't have access to these sources, you may be able

to find comparable Linux discussion forums on online services, like BBS's and CompuServe. Also available online are *Linux Journal's* Best of Technical Support columns, at `http://www.linuxjournal.com/techsup.html`.

Several businesses provide commercial support for Linux. These services allow you to pay a subscription fee that lets you call consultants for help with your Linux problems.

Keeping the following suggestions in mind will greatly improve your experience with Linux and guarantee more success in finding help.

Consult all available documentation... first! You should do this when you first encounter a problem. Various sources of information are listed in Section 1.9 and Appendix A. These documents are laboriously written for people who need help with the Linux system, like you. As mentioned above, books written for UNIX are applicable to Linux, and you should use them, too.

If you have access to Usenet news, or any of the Linux-related mailing lists, be sure to read the information there before posting. Often, solutions to common problems that are not easy to find in the documentation are well-covered in newsgroups and mailing lists. If you only post to these groups but don't read them, you are asking for trouble.

Learn to appreciate self-reliance. You asked for it by running Linux in the first place. Remember, Linux is all about hacking and fixing problems. It is not a commercial operating system, nor does it try to be one. Hacking won't kill you. In fact, it will be enlightening to investigate and solve problems yourself—you may even one day call yourself a Linux guru. Learn to appreciate the full value of hacking the system and fixing problems yourself. You shouldn't expect to run a complete, homebrew Linux system without some handiwork.

Remain calm. Nothing is earned by taking an axe—or worse, a powerful electromagnet—to your Linux box. A large punching bag or a long walk is a good way to relieve occasional stress attacks. As Linux matures and distributions become more reliable, we hope this problem will disappear. However, even commercial UNIX implementations can be tricky. When all else fails, sit back, take a few deep breaths, and return to the problem when you feel relaxed. Your mind and conscience will be clearer.

Refrain from posting spuriously. Many people make the mistake of posting or mailing messages pleading for help prematurely. When encountering a problem, do not rush immediately to the nearest terminal and post a message to one of the Linux Usenet groups. First try to resolve the problem yourself, and be absolutely certain what the problem is. Does your system not respond when switched on? Perhaps it is unplugged.

When you post for help, make it worthwhile. Remember that people who read your post are not necessarily there to help you. Therefore, it is important to remain as polite, terse, and informative as possible.

How does one accomplish this? First, you should include as much relevant information about your system and your problem as possible. Posting the simple request, “I cannot seem to get e-mail to work” will probably get you nowhere unless you include information about your system, what software you’re using, what you have attempted to do so far, and what the results were. When you include technical information, it is also a good idea to include general information about the version of your software (the Linux kernel version, for example), as well as a brief summary of your hardware configuration. But don’t overdo it—your monitor type and brand is probably irrelevant if you’re trying to configure network software.

Chapter 2

Obtaining and Installing Linux

David Bandel rewrote and revised the first section on installing Linux. Parts of the work by the following authors of the different sections on Linux distributions were also added to this first section.

Boris Beletsky wrote the Debian section. Sean Dreilinger wrote the section on Slackware. Henry Pierce wrote the section on Red Hat Linux. Evan Leibovitch wrote the section on Caldera OpenLinux. Larry Ayers wrote the section on S.u.S.E. Linux.

2.1 Generic installation.

Unlike most other operating systems, Linux can be obtained free of charge. Due to the GNU General Public License under which Linux is distributed (see Appendix C), no one can sell you a license for the software. You can use Linux at no charge and are encouraged to make it available to others.

But that doesn't mean companies aren't entitled to reimbursement for copying costs plus a profit. They may also add software that is not free that runs on the system.

This gives you the freedom to choose. If purchasing a CD-ROM is not within your budget, you may simply borrow a friend's copy or download the source from the Internet. Whether purchased from a major Linux distributor or downloaded from their FTP site (see Appendix B), you get the same operating system and the software packages that they offer. In fact, you can get more free software from one of the FTP sites than the companies can distribute on CD, due to restrictions some authors place on the distribution of their software.

2.1.1 Major Linux distributions.

An in-depth look at some of the Linux distributions begins on page 53. These distributions are: Debian, Red Hat, Caldera, Slackware, and S.u.S.E. Each section has more information on where to obtain that distribution. But remember, Linux is the kernel. The software is part of the distribution, not Linux. Most of the software is freely available and can be ported between various UNIX platforms. After taking into account what the kernel itself will support, the biggest difference comes in what the libraries (software called from within the applications) support.

Each distribution has its own installation and maintenance utilities that ease installation and system administration. Each is apparently aimed at a different audience. Any distribution will get you started and keep you running. So I recommend that you read about each distribution and talk to any knowledgeable friends. Most large cities have a Linux User Group[1] , most with experienced users, who argue at length over which distribution is the best, and why. I suggest that you listen to some of their arguments and then decide. You can also join mailing lists (I recommend joining only one at a time) and reading user posts and answers from the list gurus. As different as each distribution is, so too are the mailing lists that provide assistance. Making the right choice for yourself is important, because changing distributions generally means reinstalling from scratch.

2.1.2 Common concerns.

This section makes the assumptions that the average newcomer to Linux:

- has a computer with MS-DOS and Windows or OS/2;
- has a basic understanding of MS-DOS but not UNIX;
- knows or can find out what kind of hardware the computer has installed;
- has a desire to ”try out” Linux for whatever reason, though probably not switch to it exclusively (yet); and
- has neither a spare machine nor second disk drive available, but several hundred megabytes on an existing drive free for use.

[1] See `http://www.ssc.com/glue/` for information on SSC’s Groups of Linux Users Everywhere to find a local Linux User Group.

These assumptions are not extreme, and may even be a bit conservative. Some say that if your VCR still blinks 12:00, Linux isn't for you, but then that would leave me out as well. My VCR still blinks 12:00.

Before we begin, we must know where we are going. While it is certainly possible to get from New York to California (eventually) by striking out in almost any random direction, most of us would opt to go in a more or less direct route. So it is with installing Linux.

2.1.3 Hardware.

This section explains all of the installation steps necessary short of the actual install. Each distribution handles this preparation slightly differently. While the installs look different, they accomplish the same things and have more in common than not. All require:

- planning;
- gathering system hardware information;
- backing up your old system (optional, but strongly recommended);
- preparing Linux partitions;
- deciding on a boot loader (for dual boot systems);
- booting a Linux kernel;
- installing the kernel;
- choosing and installing software packages;
- loading the software;
- making final configuration adjustments; and
- rebooting into a running system.

Now that I've sufficiently oversimplified the process, let's go down the list. Hang on, it's not that bad when you learn from others' mistakes.

2.1.4 Planning.

I can't overemphasize this step. Any pilot will tell you that the landing is only as good as the approach. The same goes for Linux installation.

First, determine what kind of hardware you have. A checklist has been included to assist you. Be as precise as possible, but don't get carried away. For example, if you have an Ethernet card, you need to know what kind (e.g., SMC-Ultra, 3Com 3C509, etc.), base I/O (e.g., io=0x300), interrupt (IRQ 10), but not the hardware address (00 00 a6 27 bf 3c). Not all information will be needed for your hardware. If you have Windows 95 or Windows NT running, you can copy the values from the system hardware device information screen. Otherwise, consult the hardware manuals or the hardware company's Web site. Since it is important, we'll review this worksheet here.

2.1.5 System planning worksheet.

General

Processor:	Type:	386 486 Pentium PPro		
Speed (optional):				
Mfg:				
Intel AMD Cyrix				
Motherboard:	Make:	Chip Set:		
Example:	*Make: unknown*	*Chip Set: triton II*		
Mouse:	Mfg:	Type: bus PS/2 serial port		
If serial:	COM1 (ttyS0)	COM2 (ttyS1)		
Hard disk drive(s):	Type:	IDE/MFM/RLL/ESDI SCSI		
Size (list each drive):				
If SCSI Controller:	Make:	Model:		
Example:	*Make: BusLogic*	*Model: 948*		
Boot:	Linux	DOS/Windows	OS/2	Other
Disk:	Partition:	Size:	Boot:	
Disk:	Partition:	Size:	Boot:	
Disk:	Partition:	Size:	Boot:	
Disk:	Partition:	Size:	Boot:	
CD-ROM:	IDE/ATAPI SCSI	Proprietary		
Mfg:	Model:			
(Proprietary only):				

X-Windows:

Video Card:	Mfg:	Model:		
RAM:	1Mb 2Mb 4Mb 8Mb 16Mb			
Monitor:	Mfg:	Model:	Max scan rate:	

Networking:

Modem:	Mfg:	Model:		
Serial port:	COM1	COM2	COM3	COM4
	(ttyS0)	(ttyS1)	(ttyS2)	(ttyS3)
Computer hostname:		*Example: rainier*		

The following answers are only needed if using network interface card (NIC): (do not configure networking if you do not have a NIC installed)

NIC Type:	ethernet	token ring	FDDI	other
NIC Mfg:	Model:			
Network domain name:		*(Example: mountains.net)*		
IP Address:		*(Ex: 192.168.1.2)*		
Network address:		*(Ex: 192.168.1.0)*		
Netmask:		*(Ex: 255.255.255.0)*		
Broadcast address:		*(Ex: 192.168.1.255)*		
Gateway(s):		*(Ex: none or 192.168.1.1)*		
DNS(s):		*(Ex: 192.168.1.2)*		

Some of the General Section is there for future reference. Specifically, we don't need to know right now our CPU processor type. We can also do without ever knowing what chip set we have on the motherboard. But if the information is available, it is good to have.

2.1.6 Mice.

Other information, beginning with the mouse, we do need, if we expect to use the mouse. We need to know the mouse manufacturer, because different brands implement internal signal functions differently. Here, attention to detail is everything. If you have a mouse with a Microsoft brand on it, it may have a serial or PS/2 interface. Looking at the connector for the computer won't help, either. A number of computers come with mice that look like serial mice and have a serial-type connector, but are connected to the motherboard internally as a PS/2 mouse.

Read the print on the bottom of the mouse carefully before deciding. Also, if you have a mouse with three buttons, but it has a switch on the bottom which you can change between, say, Microsoft and PC systems, choose PC system. The Microsoft setting doesn't implement the middle button, which is useful in UNIX. For manufacturer, choose the switch setting, since that is the signaling protocol used. No drivers exist for a "Cutie" mouse, but do exist for the switch settings of Microsoft and Mouse System found on the bottom of the mouse.

While not specifically asked for, the only additional information you may want to add is the device through which the system accesses the mouse. Linux must know how the device is referred to. If you have a PS/2 mouse, you will normally use either `/dev/psaux`, the auxiliary port for a PS/2 pointing device, or `/dev/psmouse`, a synonym sometimes available for use. Bus mice are accessed through a file specifically created for that proprietary mouse, like `/dev/atibm` ATI bus mice, `/dev/logibm` for Logitech bus mice, `/dev/inportbm` for InPort bus mice, or their respective synonyms of `atimouse`, `logimouse`, and so on. For serial mice, if you know the MS-DOS `COM:` port, substitute `/dev/ttyS0` for `COM1:` and `/dev/ttyS1` for `COM2:`. I'll refrain from explaining the origins of the `tty` name of `ttyS0` since that will take up several paragraphs and is already explained in many UNIX references

2.1.7 Considering Hard drives and CD-ROMs.

Before you begin installation, you need to take stock of how much hard disk real estate you will dedicate to Linux versus how much you have. Deciding during installation how you want to divide up your hard disk is only asking for problems and will probably end up in lost time, lost data, and a reinstall.

Your hard drive will be one of several kinds. For our purposes, IDE, MFM, RLL, and ESDI are equivalent, and I will use the term IDE. This also encompasses EIDE, the most common interface currently in home computer systems on the market and favored for its low price.

If the hard drive has a SCSI interface, this will be evident during booting. You will need to know the make and model of the SCSI controller. The most common are the Adaptec and BusLogic controllers, but by no means are they the only ones. These also have specific models, like the AHA-1572 or BTC-958. This information is often displayed during system initialization.

To allocate space, we need to assess the hard drive size. Under OS/2, you can use the entire hard disk for OS/2, then install Microsoft Windows on the partition with OS/2 and

run Microsoft Windows under OS/2. If you have MS-DOS and Microsoft Windows, or OS/2 on your computer, Linux should have its own partition. It can be loaded on a MS-DOS partition with UMSDOS, which is not covered here. While Linux has DOS emulators and can read and even run some DOS programs, DOS cannot usually "see" what is on a Linux partition.

If you have and want to keep MS-DOS (assumed), you must determine how much space to reserve for it. Subtract this number from the hard disk total and that is what you have to work with. For now, annotate the total size of the drive(s) you have and the second number with how much to dedicate to Linux.

For your CD-ROM drive(s) you need similar information. A CD-ROM drive is either IDE/ATAPI, the most common in home systems marketed today; SCSI; or an older, proprietary drive, like those connected to sound cards. If you have an IDE or SCSI drive, so much the better. If you have a proprietary drive, you must know the make and model because Linux identifies proprietary CD-ROM drives by manufacturer and the specific drive.

2.1.8 Disk drives under Linux.

For newcomers to Linux who are familiar only with MS-DOS, and for those who come from other UNIX platforms, devices under Linux have have peculiar references. These references are used almost from the start, and some understanding of them is necessary.

Under Linux, as under any UNIX, **devices** are special files. Hard drives are treated as files and are referred to by name, as are modems, your monitor screen, and other hardware devices. UNIX treats them as files to be read from and written to. Since Linux sees them as files, they will all be located in a directory dedicated to devices. After installation you will be able to see them under the directory `/dev`, for *devices.*

Even though these devices are seen by Linux as files, they are special. They come in two "flavors," **block** and **character**, which refer to the way the device communicates, in blocks of data or individual characters. They are created automatically during installation.

These naming conventions are discussed on page 198.

2.1.9 Installing The X Window System

I've included on your worksheet information regarding your video card and monitor. While this is not absolutely necessary, most of those who come from the Microsoft Windows or OS/2 world want to install and configure a **graphical user interface** (GUI) for use. Some distributions walk you through this set up, others point you to post-installation programs. This information will be important then.

You must know the manufacturer and specific model of your video card. Some cards can be probed for RAM or chip sets, others can't. In either case, knowing how much RAM is on the card and the chips used, like the S3 or S3-Virge, is important. This information saves much time and grief. The most difficult and frustrating part of any Linux installation and setup is the X Window System.

The data for your monitor is often more difficult to obtain. If you have one of the more obscure brands of monitors, you may need to supply vertical and horizontal scan rates yourself.

◇ If in doubt, always err on the conservative side. Overdriving your system can result in damage to the monitor or video card.

We already have most of the information that we need for the mouse, the only other subsystem the X server needs. The information that Linux needs to know about your mouse is described on page 40.

2.1.10 Networking hardware.

This section is not as significant yet as the worksheet suggests. Networking is explained in detail in Chapter **??**. But if you have a **network interface card** (NIC), be it Ethernet, token ring, or other system, you must read up on the card before proceeding. This information is needed during installation to use the NIC.

During initial Linux installation, if you do not have a NIC, you can skip over some of the networking part. However, all computers must have a name under Linux. The example on the worksheet assumes you've chosen a theme, like mountains, and will name your computers after names of mountains, but any scheme your concoct is fine.

If you have a modem, you must know where it is connected. This is a serial port, `/dev/ttyS0` through `/dev/ttyS3`, corresponding to MS-DOS `COM:` ports 1–4. ISDN is treated similarly, but is normally set up post-installation with special, multiple device designations.

That finishes our worksheet and about half of the planning that we need to do. One notation that is not on our worksheet is the amount of random access memory (RAM) the system has. Linux runs happily on a system with less than 4 MB of RAM, but this has a significant impact on installation and subsequent system usage. If you have 4 MB of RAM or less, then you must follow special procedures for low memory machines where applicable. With the current low price of RAM and with few machines being sold today with less than 16MB of RAM, this generally isn't an issue. If it is, be sure to check your distribution for special instructions.

2.1.11 Planning, Part 2.

Portions of the following section, particularly the disk partitioning strategies, are highly contentious among seasoned installers, but I will give you my thoughts on it. You are welcome to deviate as you see fit. Most differences in opinion, though, come from a difference in the ultimate purpose for the system; i.e., as a workstation, Web server, News server, or other function.

2.1.12 Partitioning strategies.

Few seasoned Linux users will tell you to make one Linux native partition and one swap partition and start installation. There are several reasons for this, and I subscribe to most of them, so I have several Linux native partitions. But to me, the most compelling reason of all, is that one day you will want to upgrade, and that will require reformatting the file system(s). In fact, the Slackware distribution has, at last look, no means to even attempt an "in-place" upgrade, or any indication that it will in the future. Upgrading from kernel 0.99 to 1.2.13 required me to reformat, as did the upgrade from 1.2.13 to 2.0.0, and I suspect that another will be required for 2.2.0 (or whatever the next stable kernel is). What I don't ever want to do is lose the files that I have accumulated in my home directory. Yes, I have a backup. But keeping my `/home` directory intact is easier, especially since I moved all my special files to a subdirectory there.

◇ Another reason is that any bootable partition must be within the first 1024 cylinders of the hard drive. When any PC boots up, a sequence of events occurs which ends in the loading of the operating system. Due to limitations in the BIOS (Basic Input/Output System), until the operating system is loaded, only the first 1024 cylinders of the first or second hard disk can be accessed.

To get a feel for exactly what we're talking about, I'm going to describe a standard Linux file system and how Linux handles partitions.

Under MS-DOS, every partition is a different drive, and little distinction is made between whether that is a physical drive or a logical drive (partition). Under Linux, physical and logical drives are much less rigidly designated.

During installation, you must choose a partition as your root partition. The root partition is designated as "`/`". When we refer to "`/dev`", this is really two directories, "`/`" and "`dev`". Your Linux kernel will be located on the root partition, but can be in a subdirectory as long as that subdirectory resides on the root partition. For example, some distributions use `/boot` to hold the kernel, system map, and boot up files.

The following structure (as a minimum) will be set up on your root partition during

install:

```
/bin
/dev
/etc
/home
/lib
/lost+found
/proc
/root
/sbin
/usr
/var
```

You may see others like `/boot`, `/mnt`, `/cdrom`, `/floppy`, `/opt`, and so on, but the above are essential.

What about other partitions? Linux can use a directory name (say `/usr`) as a **mount point**. That is, the other partition on the disk (or on another disk) is mounted under it (in this case `/usr`).

If you unmount the other partition and look in the subdirectory Linux uses as a mount point, you will (or should) see nothing—no files or directories. When the other partition is mounted, you will see files and directories which are on that partition under the mount point. So if you have two drives, one with 120 MB and another with 840 MB, you can make one partition on the 120MB drive (let's say it's the root partition) and mount any partitions you have created on the 840MB drive (this could be one big partition, or several smaller partitions) under their respective mount points, one partition per mount point, creating, in effect, one, 960-MB file system.

The one restriction is that you cannot use certain directories on the root drive as mount points, because they contain files that are needed to either boot the system or mount other systems. Obviously if the command used to mount other partitions is located on another partition and you can't access that partition until you've mounted it, you'll be like the dog chasing its tail.

◇ The directories you cannot use as mount points are: `/bin`, `/dev`, `/etc`, `/lib`, `/lost+found`, `/proc`, `/root`, and `/sbin`.

A detailed description of what files are contained in these standard system directories is given on page 148.

Let's look at a small example. You are an aspiring Internet Service Provider (ISP). You

have four machines, and each has a 1-gigabyte drive. So, you decide to allocate space as follows:

```
machine A:  / = 120MB
/usr = remainder of drive (exported)
/home = 0 - mount point (mounted from B)
/var/news = 0 - mount point (mounted from C)
/var/spool/mail = 0 - mount point (mounted from D)

machine B: / = 120MB
/usr = 0 - mount point (mounted from A)
/home = remainder of drive (exported)
/var/news = 0 - mount point (mounted from C)
/var/spool/mail = 0 - mount point (mounted from D)

machine C: / = 120MB
/usr = 0 - mount point (mounted from A)
/home = 0 - mount point (mounted from B)
/var/news = remainder of drive (exported)
/var/spool/mail = 0 - mount point (mounted from D)

machine D:  (reader exercise)
```

You probably noticed that I arbitrarily assigned the root partition 120MB, and allocated the rest to whatever (`/usr`, `/home`, `/var/spool/mail` and so forth). I also didn't allocate any space to a swap partition. So, let's look at what we will likely need, understanding that "it depends," is key. I will discuss this from the perspective of a home situation with only a few users, lots of programs, and no other remarkable needs.

The best place to start is to tell you what my primary home computer looks like. I have two drives, `/dev/hda` (1.2 GB) and `/dev/hdb` (540 MB). `df` (disk free) displays

```
File system         1024-blocks  Used Available Capacity Mounted on
/dev/hda1             150259   69605    72894     49%   /
/dev/hda3             723923  615452    71075     90%   /usr
/dev/hda2             150291   93326    49204     65%   /usr/X11R6
/dev/hdb1             499620  455044    18773     96%   /home
```

You can see that I have a half-used 150-MB root (`/`) partition, a nearly full `/usr` partition, a largely used `/usr/X11R6` partition, and a large, but cramped, 500-MB /home partition. The remainder of the drive `/dev/hdb` is a swap partition.

At a realistic minimum, I would suggest reserving 80–100 MB for your root partition, about 10 MB per user on your `/home` partition, as much space as you can reserve for swap, within reason (see the next section), and the rest to `/usr`. I have a five-user system at home, but I personally have over 400 MB of the `/home` directory tied up, much of that in graphics—a photo album of family and friends. Your `/usr` partition should probably be at least 250 MB, but the minimum will depend on what you decide to install. As you can see, it can rapidly fill with over 800 MB of programs, libraries, and data. Also remember that partitions give you flexibility that you lose with one, giant partition.

2.1.13 The swap partition.

You must give thought to a swap partition. Unlike Microsoft Windows, Linux uses a dedicated swap partition for speed. Although it is possible to create a swap file, it is not recommended. Linux can use up to 128 MB of swap space. I recommend a practical minimum of 16 MB. The optimum is probably as much as you can spare between 32 and 64MB—the more, the better.

One last consideration before you decide to how best to carve up the disk. Remember that I said the BIOS cannot "see" past sector 1023 on the hard drive (about 512MB). So, the Linux kernel (a file probably called `vmlinuz` on your boot disk), or any OS kernel for that matter, must reside entirely on one of the first two disk drives (`/dev/hda` or `/dev/hdb`) and within the first 1024 sectors, or the BIOS will be unable to load it. To insure that it can, plan to make your root partition (as well as any other boot partition) fall entirely within this limitation on either the first or second hard drive.

2.1.14 Repartitioning.

At the beginning of this chapter I said I'd make a few assumptions. One was that you would want to keep your comfortable MS-DOS and Microsoft Windows operating system around. And since the computer you bought only has MS-DOS on it, it doesn't make sense to have multiple partitions, so the one drive you have is probably entirely dedicated to MS-DOS.

One way or another, then, we will have two operating systems on this computer. If you currently have nothing on your disk (lucky you), that is great, but you're not quite ready to skip ahead. Linux is comfortable wherever you put it. Your BIOS may not be capable of booting it, but once running, it will not complain if it's relegated to the fourth partition of the fourth hard drive. But MS-DOS and Microsoft Windows aren't so forgiving. They want the first drive and the first partition and may refuse to boot from any other position. I

have seen MS-DOS boot from the first partition on the second hard drive, but the first hard drive did not have any MS-DOS partitions, so MS-DOS didn't recognize the drive. The best strategy is often the path of least resistance. If at all possible, put MS-DOS on the first drive and the first partition.

A second consideration in a multiple OS situation is which operating system to load first. If you're tempted to partition the hard disk and install Linux first (reserving `/dev/hda1` for MS-DOS, then installing MS-DOS second, don't. Windows 95 is the worst offender, but Microsoft products in general will delete any previous boot loader you had installed on the master boot record (what the BIOS uses to point to bootable kernels). In fact, you may even hear this referred to as the "Microsoft virus". This is not a virus in the true sense of the word, just arrogance on the part of Microsoft, that one would only want a Microsoft operating system to boot. Linux does not cause such problems, and in fact provides a way to choose the default boot image. It also allows you to intervene during the boot process to specify which operating system to boot. This is a standard part of Linux installation procedures.

2.1.15 Backing up your old system.

Before we actually get to work on the partition table, I will walk through procedures to protect the data that you have on the hard disk. These procedures assume that you have a DOS partition. Other operating systems may or may not have a way to accomplish the same thing.

◇ The first thing that you should do is perform a complete backup. The tools that you will use work as they should. But these procedures are inherently dangerous. Any time you work with a hard disk partition table, you can easily lose all the data contained on the drive. **Back up your hard disk before you proceed.**

Once you have your disk backed up, create a boot floppy disk for the system. You can either use the MS-DOS command

```
C:> format a:  /s
```

which formats the floppy and puts the required system files on it, or, using a formatted disk, issue the command

```
C:>sys a:
```

Once you have created a boot floppy and tested it to insure that it works, copy the following files from your MS-DOS system to the boot floppy: `FDISK.EXE`,

SCANDISK.EXE, and SYS.COM. Also copy the file RESTORRB.EXE from a Linux distribution CD or Linux FTP archive. (See Appendix B).

Run a defragmentation program on your DOS drive to defragment and group the files together at the front of the disk. If defragmeter encounters any errors, you need to run SCANDISK.EXE to fix the problems. Once you have defragmented the disk and ensured that the files are compressed toward the front of the drive (as indicated in the graphical portrayal of your disk), you're ready to run FIPS.EXE to shrink the MS-DOS partition.

2.1.16 FIPS.EXE

On your Linux distribution CD (or an Internet distribution site), you'll find a copy of FIPS.EXE, which can shrink the MS-DOS partition. Note that FIPS.EXE only works for MS-DOS partitions. If you have other partitions that you need to shrink, the program Partition Magic may help, but is not free. Copy FIPS.EXE to your boot floppy and reboot using this floppy. This accomplishes two things: it insures that the boot floppy works, and insures that you are booted into MS-DOS Real Mode and are not running Microsoft Windows.

At the A:> prompt, type FIPS (upper or lower case). You will be greeted and asked which drive you want to operate on (if you have more than one). Select the drive to shrink. Once you confirm your choice, let FIPS.EXE make a copy of your boot and root sectors to the floppy in case something untoward happens.

You will then be asked if all of the free space on your partition should be used to create a second partition. If you say, "yes," you will not have any free space on the MS-DOS partition to save data to, so say, "no." You will then be able to alter the amount of space allocated between the first and second partitions. Note that if you didn't properly defragment your drive, you won't have much to work with on the second partition. Also, if you use MS-DOS mirroring software, a file is created at the very end of the partition, and FIPS.EXE tells you that you have no space to create a second partition. Exit and correct the problem by deleting the MIRROR.FIL file, then restart FIPS.EXE.

You can edit and re-edit the table until you are satisfied. Once you are happy with the distribution of space between the partitions, confirm your changes and write out the table.

Once FIPS.EXE has finished, remove the boot floppy and reboot your computer. In this example, we'll destroy and recreate the second partition during installation to create at least two partitions for Linux: a swap partition and a Linux native partition. But you can create as many as you like.

2.1.17 Preparing to boot Linux.

In order to install Linux, we must begin by booting the Linux kernel. This is accomplished in exactly the same manner as if you wanted to reload MS-DOS: we need a boot disk. But most distributions come only with a CD-ROM, and even if we had a running Linux system, the command to create boot disks for Linux is different than for MS-DOS. If you bought a new computer with a bootable CD-ROM, some distributions allow you to boot in this manner. But we'll go through the process of creating a boot disk for the rest of us.

2.1.18 Creating a Linux boot disk under DOS.

Each distribution CD contains a MS-DOS program that allows you to write a raw disk image to a formatted floppy disk. You must have a high density floppy, and some distributions require this to be a 3.5-inch, 1.44 Mb floppy. Insert the floppy in the drive. On the CD (or on your disk drive if you downloaded it) find `RAWRITE2.EXE` (you may have the older `RAWRITE.EXE`).

Then `cd` to the directory that has the disk image(s) that you need to boot with. There may be only one, or many which are configured for different hardware. You will have to consult the distribution documentation. Running `RAWRITE2.EXE` with no arguments results in your having to answer two questions: the path name of the disk image file to write. and the destination disk drive, either `A:` or `B:`. To shortcut the prompts with `RAWRITE.EXE` or `RAWRITE2.EXE`, issue the arguments on the MS-DOS command line

`C:> rawrite` *diskimage drive*

Repeat this step for any additional disk images your system needs.

◇ If you can check the floppy disks with `SCANDISK.EXE` and do a surface scan before writing the images to the floppy, you may save yourself some time later. Most initial install failures come from boot disks that are bad, and `RAWRITE2.EXE` doesn't verify the disks.

◇ This is also true if you create boot disks under Linux. The `badblocks(1)` manual page describes how to check disks for errors.

Label the disks that you create for future use.

2.1.19 Creating a Linux boot disk under Linux.

If you have an operational Linux system; for example, if you upgrade and want to create the disk images with Linux, you change to the directory with the disk images and issue the command

```
# dd if=diskimage of=boot floppy device bs=512 conv=sync ; sync
```

Substitute the disk image name for *diskimage* and the correct floppy device (almost always `/dev/fd0`), and repeat for each disk that you need. The `dd` arguments are: `if` for input file; `of` for output file, and here we want to use the floppy device; `bs` for block size, in this case 512 bytes; `conv=sync` ensures that the output file is exactly the same size as the input file. The trailing "`sync`" insures that we flush the buffers to disk immediately.

An alternate method that works, though will often be shunned by "real" Linux administrators, is the `cp` (copy) command

```
cp diskimage boot floppy device ; sync
```

Again, substitute the disk image file name for *diskimage*, the correct *boot floppy device*, and repeat the step for each disk that you need. You may receive a message asking if you want to replace the boot floppy device with *diskimage*. Obviously this won't happen, since the floppy diskette is not a true file but a device, but `cp` doesn't pay attention to that detail. Just say, "yes," if you are asked.

With the Linux installation boot disks in hand, you're ready to install our system. Most distributions invoke `fdisk`, the Linux version, so you can create a native Linux partition and a swap partition. The install programs continue by creating the file system (the equivalent of formatting a MS-DOS disk) for both the Linux and swap partitions, and initialize the swap partition and mount the Linux partition.

One question that you will be asked is whether you want to check your hard disk for bad blocks. If you are using a SCSI drive, answer "no." SCSI drives have built-in error checking and correcting. IDE and similar drives don't have this and need to map out bad blocks. If you have an older drive you want to do this. If you say, "yes," the installation program will invoke the `badblocks` program to maps out all of the bad blocks it finds. This takes time. If in doubt, say, "yes."

2.1.20 Partitioning the hard disk: `fdisk` and `cfdisk`.

Every operating system, be it MS-DOS and Microsoft Windows, or Linux, has its own version of `fdisk`. If you want to create a partition for use by MS-DOS, use the MS-DOS version, `FDISK.EXE`, to create the partition and write the table. If you are going to create a partition for Linux, you must create it with the Linux version, `fdisk`.

Under Linux, two disk partitioning programs are available: the original `fdisk`, and a friendlier `cfisk`. The difference between the two is that in `fdisk` you issue all commands via the keyboard with letters and numbers. With `cfdisk`, you use the arrow keys

to highlight the options you want, and press Enter to execute the command. The only time you use anything but the arrow and Enter keys is when you specify a number for the size of the partition.

For starters, all Linux boot disks are created essentially equal. Reboot the computer with the boot floppy in the boot drive. You will be greeted with a screen with some instructions and a prompt

```
LILO boot:
```

and a flashing cursor. If you use the Tab key, you should see a list of names. The names differ depending on the distribution, but look for one that says "rescue" or "expert." The "install" label starts the installation program after loading the kernel, so if you want to let the installation program walk you through the partitioning and filesystem initialization process, you can use the "install" label; otherwise, choose a different label. You may also need to provide Linux some boot parameters. For our purposes, this should not be necessary, but you'll soon find out if this is the case.

Enter a label name and press /keyReturn. When the Linux kernel finishes the bootup process, you may be presented with any of a number of prompts, depending on the distribution. If you have a shell prompt, like the pound sign (#) or a dollar sign ($), you're where you need to be. If not, try presssing Alt-F2 or Alt-Shift-F2. You should be able to activate one of the system's virtual consoles.

Once you have a prompt (you should not need to log in), you will be working as "root" (more on this in Chapter 4). Enter the command

```
# fdisk
```

If an error is returned, try `cfdisk`. This is the disk partition utility. It defaults to `/dev/hda`, so if you need to work on the second hard drive, use the command

```
# fdisk /dev/hdb
```

In `fdisk`, press m to see a menu. The commands you will use are: n to create a new partition; d to destroy a partition; t to change the partition type (83 is Linux Native, 82 is Linux Swap); p prints to the screen the partition information currently in memory (not what's on the disk); w writes the partition table to disk; and q quits.

◇ Until you issue the w command, you are not committed and can make changes or quit without making any changes.

◇ Pay attention to prefixes and suffixes of the partition size. With the partition size you need to specify "+" if the size will be other than the ending partition number, and a suffix of "k" or "M" (case does not matter) to specify KB or MB.

One final note on partitions: you can create up to four primary partitions. If you need more than four partitions, you will create three primary partitions and then extended partitions. The extended partition numbers begin with 5, so you may have `/dev/hda1`, `/dev/hda2`, `/dev/hda3`, `/dev/hda5`, and `/dev/hda6` if you need five partitions.

As a final check before you write the partition table, ensure that your partitions do not overlap. As long as the start and end segments don't overlap with any other start and end segments, you can be sure the partition boundaries are okay. A beginning number may be listed as 1024 for partitions with numbers starting higher than that. For now, just consider that a reminder that the BIOS will not be able to read (or boot from) that partition.

`cfdisk` does exactly the same thing as `fdisk`, but displays on the screen the state of the partition table in memory (but not on the disk) at all times. Use the Up and Down arrow keys to select a partition to work on, and the Right and Left arrow keys to select the action to be performed. Then press `Enter` to perform the action. You will have to input numbers for the size you want to make the partition, but all information is given on the screen, just follow the instructions. `cfdisk` defaults to `/dev/hda`, so you must to give it the argument `/dev/hdb` if you want to change the partition table on a second disk drive. Remember to write the table before you quit. This is the hardest part of `cfdisk`. It doesn't ask for confirmation before exiting. So select Write and press `Enter` before you select Quit and press `Enter`.

2.2 Linux distributions.

You are now faced with the task of deciding which particular distribution of Linux suits your needs. Not all distributions are alike. Many of them come with just about all of the software you'd need to run a complete system—and then some. Other Linux distributions are "small" distributions intended for users without copious amounts of disk space. Many distributions contain only the core Linux software, and you are expected to install larger software packages, such as the X Window System, yourself. (In Chapter **??** we'll show you how.)

The Linux *Distribution HOWTO* (see Appendix A) contains a list of Linux distributions available on the Internet as well as by mail order.

If you have access to USENET news, or another computer conferencing system, you might want to ask there for personal opinions from people who have installed Linux. Also, *Linux Journal* maintains a table of features comparing Linux Distributions and periodically publishes software reviews of distributions (check `http://www.linuxjournal.com/selected.html` for on-line versions of the

table and articles). Even better, if you know someone who installed Linux, ask them for help and advice. There are many factors to consider when choosing a distribution; however, everyone's needs and opinions are different. In actuality, most of the popular Linux distributions contain roughly the same set of software, so the distribution you select is more or less arbitrary.

2.3 Debian GNU/Linux.

This section on Debian GNU/Linux was written by Boris Beletsky.

2.3.1 Debian GNU/Linux installation features.

Dependencies:	yes
Install boot methods:	floppy
Install methods:	CD, hard disk, NFS, FTP
System initialization:	Sys V `init`
Ease of installation:	challenging
Graphical administration tools:	no
Installation utility:	`dselect`
Package maintenance utility:	`dselect/dpkg`

2.3.2 Getting floppy images.

If you have fast, cheap Internet access, the best way to get Debian is via anonymous FTP (see Appendix B). The home ftp site of Debian is located at `ftp.debian.org` in the `/pub/debian` directory. The structure of Debian archive is described in the table on page 55.

For a base installation of Debian you need about 12 megabytes of disk space and some floppies. First, you need boot and driver floppy images. Debian provides two sets of boot floppy images, for 1.2 and 1.44 Mb floppy disks, and one set of the base images which work with either type of floppy. Check what floppy drive your system boots from, and download the appropriate disk set.

Choose the appropriate floppy set for your hardware from the table on page 55 and write the images to floppy as described on page 50.

Directory	Contents
`./stable/`	Latest stable Debian release.
`./stable/binary-i386`	Debian packages for Intel i386 architecture.
`./stable/disks-i386`	Boot and root disks needed for Debian installation.
`./stable/disks-i386/current`	The current boot floppy set.
`./stable/disks-i386/special-kernels`	Special kernels and boot floppy disks, for hardware. Configurations that refuse to work with our regular boot floppies.
`./stable/msdos-i386`	DOS short file names for Debian packages.

Table 2.1: Debian GNU/Linux archive structure.

File Name	Label	Description
`rsc1440.bin`	Rescue Floppy	
`drv1440.bin`	Device Drivers	
`base-1.bin`	Base 1	Floppy set for systems with 1.44MB
`base-2.bin`	Base 2	floppy drive and at least 5MB RAM.
`base-3.bin`	Base 3	
`base-4.bin`	Base 4	
`base-5.bin`	Base 5	
`root.bin`	Root Disk	
`rsc1440r.bin`	Rescue Floppy	Optional Rescue Disk image for low memory systems (less then 5MB of RAM).
`rsc1200r.bin`	Rescue Floppy	
`drv1200.bin`	Device Drivers	
`base-1.bin`	Base 1	Floppy set for systems with
`base-2.bin`	Base 2	1.2MB floppy drive.
`base-3.bin`	Base 3	
`base-4.bin`	Base 4	
`root.bin`	Root Disk	

Table 2.2: Debian GNU/Linux installation floppies.

2.3.3 Downloading the packages.

To install and use Debian, you need more than the base system. To decide what packages you want, download the file `Packages` from:

```
ftp://ftp.debian.org/pub/debian/stable/Packages
```

This file is a current list of Debian packages available in the stable Debian distribution. The file comes in a special format; every package has its own entry separated by a blank line. Each package's information is broken up into fields. The table on page 65 describes the fields and their possible values. It should give you an idea of how to build your personal download list. When you have the list of the packages you want, you need to decide how to download them. If you are an experienced user, you may want to download the `netbase` package—and SLIP and PPP, if necessary—so you can download packages later, via Linux. Otherwise, you can download all of the packages with your current operating system and install them later from a mounted partition.

2.3.4 Booting from floppies and installing Debian GNU/Linux.

The Rescue floppy. Place the Rescue floppy in the boot drive and reboot. In a minute or two, you should see a screen introduce the Rescue floppy and the `boot` prompt.

◇ It's called the *Rescue* floppy because you can use it to boot your system and perform repairs if there is a problem that makes your hard disk unbootable. Save this floppy after you install the system.

You can do two things at the `boot:` prompt: press the function keys `F1` through `F10` to view a few pages of helpful information, or boot the system. If you have any hardware devices that Linux doesn't access correctly at boot time, you may find a parameter to add to the boot command line in the screens you see by pressing `F3`, `F4`, and `F5`.

◇ If you add parameters to the boot command line, be sure to type the word "`linux`" and a space before the first parameter. If you simply press `Enter`, that's the same as typing "`linux`" without any special parameters.

If this is the first time you're booting the system, press `Enter` and see if it works correctly. It probably will. If not, you can reboot later and look for any special parameters that inform the system about your hardware.

Once you press `Enter`, you should see the messages

```
Loading...
Uncompressing Linux...
```

then there is a page or so of cryptic information about the hardware in the system. There may be many messages in the form of, "`can't find` *something*," "*something* `not present`," "`can't initialize` *something*," or even "`this driver release depends on` *something*," Most of these are harmless. The installation boot disk is built to run on computers with many different peripheral devices. Obviously, no computer will have every possible peripheral device, and the operating system may emit a few complaints while it looks for peripherals you don't own. You may also see the system pause for a while. This happens if it is waiting for a device to respond that is not present on your system. If you find that the time it takes to boot the system unacceptably long, you can create a custom kernel after you install the system which doesn't have the drivers for non-existent devices.

Low memory systems. If your system has 4MB of RAM, you may see a paragraph about low memory and a text menu with three choices. If your system has enough RAM, you won't see this at all, and you'll go directly to the color or monochrome dialog box. If you get the low-memory menu, you should go through its selections in order. Partition your disk, activate the swap partition, and start the graphical installation system. The program that is used to partition your disk is called `cfdisk`, and you should see the manual page for `cfdisk` and the instructions on page 51 for assistance.

`cfdisk` is used to create a Linux Swap partition (type 82) on the hard drive. You need the swap partition for virtual memory during installation, because the procedure likely uses more memory than you have physical RAM for. Select the amount of virtual memory that you intend to use once your system is installed. It is exactly equal to the amount of disk space required. Sixteen megabytes is probably the smallest practical amount, but use 32 megabytes if you can spare the disk space, and 64 megabytes if the disk is large enough and you won't miss the space.

The color or monochrome dialog box. Once the system finishes booting, you should see the color or monochrome dialog box. If your monitor displays black and white (monochrome), press `Enter` and continue with the installation. Otherwise, use the arrow key to move the cursor to the `Color` menu item and then press `Enter`. The display should change from black and white to color. Press `Enter` again to continue with the installation.

The Main Menu You may see a dialog box that says,

```
The installation program is determining the current state
of your system.
```

On some systems, this message flashes by too quickly to read. It is displayed between steps in the installation process. The installation program checks the state of the system after each step. This allows you to restart the installation without losing the work that you have already done, if you halt the system in the middle of the installation. If you need to restart an installation, you will be prompted to select color or monochrome again, configure the keyboard, reactivate the swap partition, and remount any disks that have been initialized. Any other installation on the system will be saved.

During the entire process, you are presented with the main menu. The choices at the top of the menu change to indicate your progress in installing the system. Phil Hughes wrote in *Linux Journal* that you could teach a chicken to install Debian. He meant that the installation process was mostly just pecking at the Enter key. The first choice on the installation menu is the next action you should perform according to what the system detects you have already done. It should say Next, and, at this point, the next item should be

```
Configure the Keyboard
```

Configuring the keyboard. Make sure that the highlight is on the Next item, and press Enter for the keyboard configuration menu. Select a keyboard that conforms to the layout used for your national language, or select something close to it if the keyboard layout you want isn't shown. After installation you can select a keyboard layout from a wider range of choices. Move the highlight to the keyboard selection and press Enter. Use the arrow keys to move the highlight—they are in the same place on all national language keyboard layouts and are independent of the keyboard configuration.

The shell. If you are an experienced UNIX or Linux user, press LeftAlt and F2 in unison for the second virtual console. That's the Alt key on the left-hand side of the Space bar and the F2 function key. You'll see a separate window running a Bourne shell clone called ash. At this point, the root file system is on the RAM disk, and there is a limited set of UNIX utilities available for your use. You can see what programs are available with the command

```
ls /bin /sbin /usr/bin /usr/sbin
```

◇ The shell and commands are there only in case something goes wrong. In particular, you should always use the menus, not the shell, to activate your swap partition, because the

menu software can't detect whether you've done this from the shell. Press LeftAlt-F1 to get back to menus. Linux provides up to 64 virtual consoles, but the Rescue floppy only uses a few of them.

◇ **Last chance!** Have you backed up your disks? Here's your first chance to wipe out all of the data on your disks, and your last chance to save your old system. If you haven't backed up all of your disks, remove the floppy from the drive, reset the system, and create a backup.

Partition your hard disks. If you have not already partitioned your disks for Linux Native and Linux Swap file systems, the menu item `Next` will be

```
Partition a Hard Disk
```

If you have already created at least one Linux Native and one Linux Swap disk partition, the `Next` menu selection will be

```
Initialize and Activate the Swap Disk Partition
```

or you may even skip that step if your system has little RAM and the installation software asked you to activate the swap partition as soon as the system started. Whatever the `Next` menu selection is, you can use the down-arrow key to select

```
Partition a Hard Disk
```

The `Partition a Hard Disk` menu item presents you with a list of disk drives you can partition and runs the `cfdisk` program (see page 51), which allows you to create and edit disk partitions. You must create at least one Linux (type 83) disk partition.

Your swap partition will be used to provide virtual memory for the system and should be between 16 and 128 megabytes in size, depending on how much disk space you have and how many large programs you want to run. Linux will not use more than 128 megabytes of swap, so there's no reason to make your swap partition larger than that. A swap partition is strongly recommended, but you can do without one if you insist and system has more than 16 Mb of RAM.

`Initialize and Activate the Swap Disk Partition`. This is the `Next` menu item after you create one disk partition. You have the choice of initializing and activating a new swap partition, activating a previously initialized partition, and doing without a swap partition. It's always permissible to re-initialize a swap partition, so select `Initialize and Activate the Swap Disk Partition` unless you are sure

that you know what you are doing. This menu choice will give you the option to scan the entire partition for unreadable disk blocks caused by defects on the surface of the hard disk platters. This is useful if you have MFM, RLL, or older IDE disks, and checking the disk never hurts. Properly working SCSI disks don't need to be scanned. They have their own internal mechanism for mapping out bad disk blocks.

The swap partition provides virtual memory to supplement the RAM in your system, and it's even used while the system is being installed. That's why we initialize it first.

Initialize a Linux disk partition. At this point, the `Next` menu item should be

```
Initialize a Linux Disk Partition
```

If it isn't, you haven't completed the disk partitioning process, or you haven't made one of the menu choices dealing with your swap partition.

You can initialize a Linux disk partition, or alternately you can mount a previously initialized partition.

◇ The boot floppies will not upgrade an old system without removing the files—Debian provides a different procedure than using the boot floppies for upgrading existing Debian systems. Thus, if you are using old disk partitions that are not empty, you should initialize them, which erases all of the files. You must initialize any partition that you created in the disk partitioning step. About the only reason to mount a partition without initializing it at this point would be to mount a partition upon which you have user files, like `/home`, that you don't want deleted.

Select the `Next` menu item, to initialize and mount the root (the "`/`" directory) disk partition. The first partition you mount or initialize, after the swap partition, if you're using it, is the partition mounted as root. You will be offered the choice to scan the disk partition for bad blocks, as when you initialized the swap partition. It never hurts to scan for bad blocks. Keep in mind that this step can take 10 minutes or more if you have a large disk.

Install the base system. Once you've mounted the root partition, the `Next` menu item will be

```
Install the Base System
```

unless you already performed some of the installation steps. You can use the arrow keys to select the menu items to initialize or mount disk partitions if you have additional partitions to set up. If you have created separate partitions for `/var`, `/usr`, or other file systems, you should initialize and mount them now.

There will be a pause while the system looks for a local copy of the base system. This search is for CD-ROM installations and will not succeed. You are then offered a menu of drives from which to read the base floppies. Select the appropriate drive. Feed in the Base 1, Base 2, Base 3, and Base 4 floppies—and Base 5 if you are using 1.2MB floppies—as requested by the program. If one of the base floppies is unreadable, you need to create a replacement floppy and feed all five floppies into the system again. After the floppies have been read, the system installs the files. This can take ten minutes or more on a slow system.

Install the operating system kernel. At this point, the `Next` menu item should be

```
Install the Operating System Kernel
```

Select it, and you will be prompted to select a floppy drive and insert the Rescue floppy. This copies the kernel onto the hard disk. This kernel is used later to create a custom boot floppy for your system and make the hard disk bootable without a floppy.

Install the device drivers. Select the menu item to install the device drivers. You will be prompted to insert the Device Drivers floppy, and the drivers will be copied onto your hard disk. Select the

```
Configure Device Drivers
```

menu item and look for devices which are on your system. Configure those device drivers, so they will be loaded whenever your system boots.

There is a menu selection for PCMCIA device drivers, but you do not need to use it. After installation you can install the `pcmcia-cs` package. This detects PCMCIA cards automatically and configures those it finds. It also recognizes cards that are hot swapped when the system is running—they will all be configured as they are plugged in, and de-configured when unplugged.

Configure the base system. At this point the system read in all of the files that make up a minimal Debian system, but you must perform some configuration before the system will run. Select

```
Configure the Base System
```

This asks you to select your time zone. Look for your time zone or region of the world in the menu, and type it at the prompt. This may lead to another menu where you can select more specific information.

Next, you are asked if your system clock should be set to Greenwich Mean Time (GMT) or local time. Select GMT if are running only Linux or another UNIX on your system. Select local time if you use another operating system like MS-DOS or Microsoft Windows. UNIX systems keep GMT time on the system clock and use software which converts it to the local time. This allows them to keep track of daylight savings time and leap years, and even allows users who are logged in from other time zones to individually set the time zone on their terminal. If you run the system clock on GMT and your locality uses daylight savings time, the system adjusts for daylight savings time properly on the days it starts and ends.

Configure the network. You must configure the network even if you don't have one, but you only have to answer the first two questions:

```
What is the name of your computer?
Is your system connected to a network?
```

If you are connected to a network, check with your system administrator or ISP vendor if you don't know the following information:

- your computer's host name;
- your computer's or ISP's domain name;
- your computer's IP address;
- the netmask to use with your network;
- the IP address of your network;
- the broadcast address to use on your network;
- if your network has a gateway, the IP address of the default gateway system to which you should route packets;
- the system on your network to use for Domain Name Service (DNS); and
- whether you connect to the network using Ethernet.

The program will guess that the network IP address is the bitwise AND of your system's IP address and netmask. It will guess that the broadcast address is the bitwise OR of your system's IP address with the bitwise negation of the netmask. It will guess that

your gateway system is also your DNS server. If you can't find any of these answers, use the system's guesses—if necessary, you can alter them after installation by editing the `/etc/init.d/network` file.

Make the hard disk bootable. If you choose to make the hard disk boot directly to Linux, you are asked to install a master boot record. If you aren't using a boot manager (this is probably the case if you don't know what a boot manager is), answer "yes" to this question. The next question is whether you want to boot Linux automatically from the hard disk when you turn on the system. This sets Linux to be the bootable partition—the one that will be loaded from the hard disk. If you answer "no" to this question, you can set the bootable partition later using the MS-DOS `FDISK.EXE` program, or the Linux `fdisk` or `activate` programs.

Make a boot floppy. You should make a boot floppy even if you intend to boot the system from the hard disk. The reason for this is that it's possible for the hard disk bootstrap to be installed incorrectly. A boot floppy will almost always work. Select

```
Make a Boot Floppy
```

from the menu and feed the system a blank floppy as directed. Make sure that the floppy isn't write protected. The software attempts to format and write it. Mark this diskette the "Custom Boot" floppy and write-protect it once it has been written.

The moment of truth. This is what electrical engineers call the "smoke test"—what happens when you power up a new system for the first time. Remove the floppy disk from the floppy drive and select

```
Reboot the System
```

from the menu. If the Linux system doesn't start up, insert the Custom Boot floppy you created in the previous step and reset the system. Linux should boot. You should see the same messages as when you first booted the installation boot floppy, followed by some new messages.

Add a user account and password. After you've added logins, (Chapter 4 discusses this in some detail), you are dropped into `dselect`, the Debian package management program.

◇ You should read the tutorial before attempting to install packages with `dselect`.

`dselect` allows you to select the packages that you want installed on your system. The Debian package management software is described in detail starting on page 64. If you have a CD-ROM or hard disk with the additional Debian packages or are connected to the Internet, you may want to read that section now. Otherwise, exit `dselect`. You can use the package management software after you have transferred the Debian package files to your system.

◇ You must be the superuser (root) to use `dselect`.

◇ If you install the X Window System and do not use a US keyboard, read the X11 Release note for non-US keyboards.

Log in. After you exit `dselect`, you are at the `login:` prompt. Log in using the personal login and password you selected. Your system is ready to use.

2.3.5 Running Debian GNU/Linux.

This section describes the Debian packaging system and Debian-specific utilities. The Debian/GNU Linux `Packages` file format is shown in the table on page 65.

Debian distributions come in archives called **packages**. Every package is a collection of files (programs, usually) that can be installed using `dpkg` or `dselect`. In addition, the package contains some information about itself that is read by the installation utilities.

Package classifications. The packages that are included with Debian GNU/Linux are classified according to how essential they are (priority) and their functionality (section).

The *priority* of a package indicates how essential or necessary it is. Debian GNU/Linux classifies all packages into four different priority levels:

Required. These packages must be installed for the system to operate correctly and have been installed as part of the base system.

◇ Never remove a required package from the system unless you are absolutely sure of what you are doing. This bears repeating: **Never, never, never remove a required package from the system unless you are absolutely sure of what you are doing.** It is likely that doing so will render your system completely unusable.

Required packages are abbreviated in `dselect` as `Req`.

Package	**Package Name**	
Priority	**Package Importance**	
	Required	Should be installed for proper system operation.
	Important	Not required but important.
	Optional	Not necessary but useful.
	Extra	Package may conflict with other packages with higher priorities.
Section	**General Category**	
	Base	Base system.
	Devel	Development tools.
	X11	Packages for the X Window System.
	Admin	Administration utilities.
	Doc	Documentation.
	Comm	Various communication utilities.
	Editors	Various editors.
	Electronics	Electronics utilities.
	Games	Games. (You knew that, didn't you?)
	Graphics	Graphics utilities.
	Hamradio	Ytilities for Internet radio.
	Mail	Email clients and servers.
	Math	Mathematics utilities. (Like calculators, etc.…)
	Net	Various tools to connect to the network (usually TCP/IP).
	News	Servers and clients for Internet news (NNTP).
	Shells	Shells, such as `tcsh` and `bash`.
	Sound	Any sound application (like audio CD players).
	TeX	Anything that can read, write, and convert TEX.
	Text	Applications to manipulate texts (like `nroff`).
	Misc	Everything else that doesn't fit above.
Maintainer	Name of the person who maintains the package and his or her e-mail address.	
Version	Version of the package in the format *upstream-version–debian-version*.	
Depends	A list of other packages upon which the current package depends and will not function without.	
Recommends	Another level of package dependencies—it is strongly recommended the packages listed in this field be installed if this package is to be used.	
Suggests	Packages listed in this field may be useful to the packages this entry describes.	
Filename	File name of the package via FTP or CD-ROM.	
MS-DOS-Filename	File name of the package in short DOS format.	

Important. Important packages are found on almost all UNIX-like operating systems. These packages include `cron`, `man`, and `vi`.

Important packages are abbreviated in `dselect` as `Imp`.

Standard. Standard packages are packages that, more or less, comprise the "standard," character based, Debian GNU/Linux system. The Standard system includes a fairly complete software development environment and GNU Emacs.

Standard packages are abbreviated in `dselect` as `Std`.

Optional. Optional packages comprise a fairly complete system. The Optional system includes TEX and the X Window System.

Optional packages are abbreviated in `dselect` as `Opt`.

Extra Extra packages are only useful to a small or select group of people, or are installed for a specific purpose. Extra packages might include such programs as electronics and ham radio applications.

Extra packages are abbreviated in `dselect` as `Xtr`.

By default, `dselect` automatically selects the Standard system if the user doesn't want to individually select the packages to be installed.

The *section* of a package indicates its functionality or use. Packages on the CD-ROM and in FTP archives are arranged in subdirectories according to function. The directory names are fairly self-explanatory: for example, the directory `admin` contains packages for system administration and the directory `devel` contains packages for software development and programming. Unlike priority levels, there are many sections, and more may be added in the future, so we do not individually describe them in this guide.

Package relationships. Each package includes information about how it relates to the other packages included with the system. There are four package relationships in Debian GNU/Linux: conflicts, dependencies, recommendations, and suggestions.

A *conflict* occurs when two or more packages cannot be installed on the same system at the same time. A good example of conflicting packages are mail transfer agents (MTAs). A MTA is a program that delivers electronic mail to users on the system and other machines on the network. Debian GNU/Linux has two mail transfer agents: `sendmail` and `smail`.

Only one mail transfer agent may be installed at a time. They both do the same job and are not designed to coexist. Therefore, the `sendmail` and `smail` packages conflict. If you try to install `sendmail` when `smail` is already installed, the Debian GNU/Linux

package maintenance system will refuse to install it. Likewise, if you try to install `smail` when `sendmail` is already installed, `dselect` (or `dpkg`; see below) will refuse to install it.

A *dependency* occurs when one package requires another package to function properly. Using our electronic mail example, users read mail with programs called mail user agents (MUAs). Popular MUAs include `elm`, `pine`, and `emacs` RMAIL mode. It is normal to install several MUAs at once because they do not conflict. But MUAs do not deliver mail—that is the job of the MTA. So all mail user agent packages *depend* on a mail transfer agent.

A package can also *recommend* or *suggest* other related packages.

2.3.6 `dselect`.

This section is a brief tutorial on Debian `dselect`. For more detailed information, refer to the `dselect` manual at

```
ftp://ftp.debian.org/debian/Debian-1.2/disks-i386/current/dselect.beginn
```

`dselect` is simple, menu-driven interface which helps install packages. It takes you through the package installation process in the order of the on-screen menu:

```
Debian Linux dselect package handling front end.
0.  [A]ccess Choose the access method to use.
1.  [U]pdate Update list of available packages, if
possible.
2.  [S]elect Request which packages you want on your
system.
3.  [I]nstall Install and upgrade wanted packages.
4.  [C]onfig Configure any packages that are
unconfigured.
5.  [R]emove Remove unwanted software.
6.  [Q]uit Quit dselect.
```

There are two ways to select an option from the menu: choose it with arrows, or press the key of the corresponding letter in brackets.

Access. In this menu you choose the method to obtain and install the packages.

Abbreviation	Description
cdrom	install from a CD-ROM
nfs	install from an NFS server (not yet mounted)
harddisk	install from a hard disk partition (not yet mounted)
mounted	install from a file system which is already mounted
floppy	install from a pile of floppy disks
ftp	install using ftp

Update. `dselect` reads the `Packages` database (described above) and creates a database of the packages available on your system.

Select. This section of the program selects the packages. Choose your the package you want and press Enter. If you have a slow machine, the screen may clear and remain blank for 15 seconds. The first thing that appears is Page 1 of the Help file. You can view this screen by pressing "?" at any point in the Select screens, and you can page through the help screens by pressing the . (period) key.

To exit the Select screen after all of the selections are complete, press Enter. This returns you to the main screen *if* there are no problems with your selection. You must resolve those problems first. When you are satisfied with any given screen, press Enter.

Dependency conflicts are quite normal and to be expected. If you select package A and that package requires the unselected package B in order to run, `dselect` warns you of the problem and will most likely suggest a solution. If package A conflicts with package B, you must decide between them.

Install `dselect` runs through the entire 800 packages and installs the ones that are selected. You will need to make decisions during this process. It is often useful to switch to a different shell to compare, for example, an old configuration file with a new one. If the old file is called `conf.modules`, for example the new file will be called `conf.modules.dpkg-new`.

The screen scrolls by fairly quickly on faster machines. You can halt the display is by pressing Control-S and restart it with Control-Q. At the end of the run, there will be a list of any uninstalled packages.

Configure. Most packages are configured in Step 3, but anything remaining can be configured here.

Remove. Remove packages that are no longer needed.

Quit. *Au revoir.*

2.3.7 dpkg.

This is a command line tool that installs and manipulates Debian packages. It has several options that allow you to install, configure, update, remove, and perform other operations on Debian packages. You can even build your own packages. `dpkg` also allows you to list the available packages, files "owned" by packages, which package owns a file, and so on.

Installing or updating new or existing packages. Type the following command:

```
# dpkg -i filename.deb
```

where *filename* is the name of the file containing a Debian package, like `tcsh_6.06-11_i386.deb`. `dpkg` is partly interactive; during the installation it may ask additional questions, like whether to install the new version of a configuration file or keep the old version.

You may also unpack a package without configuring it by entering:

```
# dpkg --unpack filename
```

If a package depends on an uninstalled package or a newer version of a package you already have, or if any other dependency problem occurs during the installation, `dpkg` will exit without configuring it.

Configuring installed packages. If `dpkg` aborts during installation and leaves a package installed, the package is left unconfigured. The Debian packaging system requires the package to be configured to avoid dependency problems. Some packages also require configuration to work properly.

To configure a package, type:

```
dpkg --configure package
```

where *package* is the name of the package, like `tcsh`. (Notice that this is not the original name of the file from which `tcsh` was installed, which was longer, included a version number, and ended in `.deb`.)

Removing installed packages. In the Debian package system, there are two ways to eliminate packages: `remove` and `purge`. The `remove` option removes the specified package; the `purge` option removes both the specified package and its configuration files. The usage is:

```
# dpkg -r package
# dpkg --purge package
```

If there are any installed packages that depend on the one you wish to remove, the package will *not* be removed, and `dpkg` will abort with an error message.

Reporting package status. To report the status of the package (e.g., installed, not installed, or unconfigured), enter:

```
# dpkg -s package
```

Listing available packages. To list the installed packages that match some pattern, type:

```
# dpkg -l package-name-pattern
```

where *package-name-pattern* is an optional argument specifying a pattern for the package names to match, like `*sh`. Normal shell wildcards are allowed. If you don't specify the pattern, all of the installed packages are listed.

Listing files owned by packages. To list all the files owned by a particular package, simply type:

```
# dpkg -L package
```

However, this does not list files created by package-specific installation scripts.

Finding the package that owns a file. To find the package which owns a particular file, type the following command:

```
# dpkg -S filename-pattern
```

where *filename-pattern* is the pattern with which to search the package names for a match. Again, normal shell wildcards are allowed.

Summary. `dpkg` is simple to use and is preferred over `dselect` when all that you need to do is install, upgrade, or remove a small number of packages. It also has some functionality that `dselect` (an interface to `dpkg`) doesn't have, like finding which package owns a file. For the full list of options, refer to the `dpkg(8)` manual page.

2.3.8 About Debian GNU/Linux.

The Debian GNU/Linux Project was created by Ian Murdock in 1993, initially under the sponsorship of the Free Software Foundation's GNU project. Later, Debian separated from FSF. Debian is the result of a volunteer effort to create a free, high-quality UNIX-compatible operating system based on the Linux kernel, complete with a suite of applications.

The Debian community is a group of more than 150 unpaid volunteers from all over the world who collaborate via the Internet. The founders of the project have formed the organization Software in the Public Interest (SPI) to sponsor Debian GNU/Linux development.

Software in the Public Interest is a non-profit organization formed when the FSF withdrew their sponsorship of Debian. The purpose of the organization is to develop and distribute free software. Its goals are very much like those of FSF, and it encourages programmers to use the GNU General Public License on their programs. However, SPI has a slightly different focus in that it is building and distributing a Linux system that diverges in many technical details from the GNU system planned by FSF. SPI still communicates with FSF and cooperates in sending them changes to GNU software and in asking its users to donate to FSF and the GNU project.

SPI can be reached by post at:

> Software in the Public Interest
> P.O. Box 70152
> Pt. Richmond, CA 94807-0152

2.3.9 Mailing lists.

There are several Debian-related mailing lists:

`debian-announce@lists.debian.org`
: Moderated. Major system announcements. Usually about one message per month.

`debian-changes@lists.debian.org`
: Announcements of new package releases for the stable distribution. Usually several messages per day.

`debian-devel-changes@lists.debian.org`
: Announcements of new package releases for the unstable distribution. Usually several messages per day.

`debian-user@lists.debian.org`
: A mailing list where users of Debian ask for and get support. Usually about 50 packages per day.

`debian-sparc@lists.debian.org`

`debian-alpha@lists.debian.org`

`debian-68k@lists.debian.org`
: Lists for those who are involved in porting Debian software to the SPARC, DEC Alpha, and Motorolla 680x0 platforms.

There are also several mailing lists for Debian developers.

You can subscribe to those mailing list by mail or the World Wide Web. For more information, please visit `http://www.debian.org/`.

2.3.10 Bug tracking system.

The Debian project has a bug tracking system that handles bug reports by users. As soon as the bug is reported, it is given a number and all of the information provided on the particular bug is stored in a file and mailed to the maintainer of the package. When the bug is fixed, it must be marked as done (“closed”) by the maintainer. If it was closed by mistake, it may be reopened.

To receive more information on the bug tracing system, send e-mail to `request@bugs.debian.org` with `help` in the body of the message.

2.3.11 Debian Acknowledgments.

Many thanks to Bruce Perens and the other authors of Debian related materials that were used to write this chapter. Thanks also to Vadik Vygonets, my beloved cousin, who helped me quite a bit. Lastly, thanks are also due to the members of Debian community for

their hard work. Let's hope that Debian GNU/Linux becomes even better.

2.3.12 Last note.

Debian GNU/Linux changes very quickly, and many facts can change more quickly than this book. The source text of this section is updated regularly. You can find it at

`http://www.cs.huji.ac.il/~borik/debian/ligs/`

2.4 Red Hat Linux.

This section on Red Hat Linux was written by Henry Pierce.

2.4.1 Red Hat Linux installation features.

Dependencies:	yes
Install boot methods:	CD, floppy,loadin (from DOS)
Install methods:	CD, hard disk, NFS, FTP, SMB
System initialization:	Sys V init
Ease of installation:	easy
Graphical administration tools:	numerous
Installation utility:	install script that calls RPM
Package maintenance utility:	RPM, glint

2.4.2 The RPM package management system.

Red Hat Linux's RPM package management system manages software by defining how a software package is built for installation and collects information about its components and installation methods during the build process. A RPM package has an organized packet of data in the header of *package*`.rpm` which can be added to a database that describes where the package belongs, what supporting packages are required, whether the required packages are installed, and provides a means to determine software dependencies.

RPM gives system administrators the ability to: upgrade individual components or entire systems while preserving the configuration of the system or package; query the database for the location of files, packages, or related information; perform package verification so packages can be installed properly, or at all; keep source packages "pristine" (provide the package author's original source and second-party patches separately) so that

porting issues can be tracked. Because RPM does this, you can install, upgrade, or remove packages with a single command line in text mode or a few clicks of the mouse in X Package Management Tool. Examples of using RPM from the command line are:

```
# rpm --install package.rpm # this installs package
# rpm --upgrade package.rpm # this upgrades package
# rpm --erase package.rpm # this removes or erases package
```

Package naming conventions. A properly built *package*`.rpm` has the following characteristics: its name identifies the package, the version, the build revision, the architecture, and the extension `.rpm`, which identifies it as a RPM package.

Take, for example, `bash-1.14.7-1.i386.rpm`. The name itself contains useful information: the package is `bash` (the Bourne Again SHell), it is version 1.14.7, and it is build 1 of the current version for Red Hat Linux. It was built for Intel or compatible 80386 or higher CPUs, and it is in RPM format. So, if you see a package named `bash-1.14.7-2.i386.rpm`, you know that it is the second build of `bash` version 1.14.7, and probably contains fixes for problems of the previous build and is more current. While the internal organization of a `*.rpm` file is beyond the scope of this section, a properly built package contains an executable file, any configuration files, the documentation (at least manual pages), any miscellaneous files directly related to the package, a record of where the packages files are to be installed, and a record of any required packages. After successful installation, information about the package is registered in the system's RPM database. A more thorough discussion of RPM package management system may be found in the RPM HOWTO (see Appendix A). It is also available at

```
http://www.redhat.com/support/docs/rpm/RPM-HOWTO/RPM-HOWTO.html
```

2.4.3 A note about upgrading Red Hat Linux.

Only upgrades from version 2.0 of Red Hat Linux and onward are supported due to major changes in Linux's binary format. Otherwise, upgrades can be performed from the same installation methods of CD-ROM, NFS, FTP, and hard disk. As of Red Hat Linux version 4.0, the upgrade option is incorporated into the Boot diskette instead of a separate program. If you upgrade from v2.1 to v3.0.3 and want to upgrade again to version 4.0, you need to create the Boot diskette instead of looking for an upgrade script, the same as Red Hat 4.*x* installation from scratch. This method does not reformat your partitions nor delete your configuration files.

2.4.4 Creating the installation floppies.

To create an Installation Floppy Kit, you need the following:

1. The Red Hat Boot diskette image, `boot.img`, which is available at:

   ```
   ftp//ftp.redhat.com/pub/redhat/current/i386/images/boot.img
   ```

   or in the `images` directory of a Red Hat CD-ROM.

2. The Red Hat Supplemental diskette image, `supp.img`, which is available at

   ```
   ftp://ftp.redhat.com/pub/redhat/current/i386/images/supp.img
   ```

   or in the `images` directory of a Red Hat CD-ROM. This diskette is required if your method of installation is not CD-ROM based, or you need PCMCIA support for any device, like a CD-ROM on a laptop, to install properly. This diskette can also be used with the Boot diskette as an emergency start disk for an installed system.

3. The program `RAWRITE.EXE`, which is available at:

   ```
   ftp://ftp.redhat.com/pub/redhat/current/i386/dosutils/rawrite.exe
   ```

   or in the `DOS` directory of a Red Hat CD-ROM.

4. MS-DOS and Windows 95 users installing Red Hat Linux for the first time on a machine that will have Linux installed as a second operating system should also obtain:

   ```
   ftp://ftp.redhat.com/pub/redhat/dos/fdips11.zip
   ```

   and unzip the files into: `C:\FIPS` if you need to free space on your hard drive.

5. An Emergency Boot diskette for an existing operating system on the target machine on which Linux will be installed as a second operating system must be created.

2.4.5 Installation media.

After you create the Installation floppies using `RAWRITE.EXE` or `dd` as described on page 50, insure that your installation method is properly set up for the Red Hat installation diskettes. For CD-ROM, NFS, FTP, and hard drive installation methods, the source must have the directory `/RedHat` on the "top level" with the directories `/base` and `/RPMS` underneath:

```
/RedHat
   |----> /RPMS (contains binary the .rpm s to be installed)
   |----> /base (contains a base system and files to set up \\
           the hard drive)
```

NFS installation. For NFS installation, you will either need a Red Hat CD-ROM on a machine (such as an existing Linux box) that can support and export an ISO-9660 file system with Rockridge Extensions, or you need to mirror one of the Red Hat distributions with the directory tree organized as described above—and of course, the proper files in each directory. The directory `/RedHat` needs to be exported to the machines on the network that are to have Red Hat Linux installed or upgraded. This machine must be on an Ethernet; you can not do an NFS install via dialup link.

Hard drive installation. Hard drive installations must have the `/RedHat` directory created relative to the root directory of the partition (it doesn't matter which partition) that will contain the Red Hat distribution obtained either from CD-ROM or an FTP site. For example, on the primary DOS partition, the path to `\RedHat` should be `C:\RedHat`. On a MS-DOS file system, it does not matter that the *package*`.rpm` names are truncated. All you need to do is make sure the `\RedHat\base` directory contains the base files from a CD-ROM or FTP site and the `\RedHat\RPMS` directory contains all of the *package*`.rpm` files from the CD-ROM or FTP site. Then you can install or upgrade from that partition. If you have an existing Linux partition that is not needed for an installation or upgrade, you can set it up as outlined here and use it.

FTP installation. To install via FTP over the Internet, all you need is the IP address of the FTP server and the root directory path for the Red Hat Linux system you wish to install. See Appendix B for a list of Linux FTP sites and mirrors. If you intend to do an FTP installation over a low-bandwidth connection (anything slower than a 128K ISDN link), it is highly recommended you copy the files to an existing MS-DOS hard

drive partition and then install from the hard drive. The total size of the packages in the `/RedHat/RPMS` directory is approximately 170MB and will take many hours to install. If something goes wrong with the installation, such as the link going down, you need to start from the beginning. If you get the files first and set up your hard drive to install Linux, it is then less work and less confusing to recover from a failed installation. You don't even need to download all of the files in `/RedHat/RPMS` to successfully install a minimal system which can grow with your needs. See the next section for details.

2.4.6 Customizing your NFS or hard drive installation.

You can customize the installation process. This is not for the faint of heart—only those already familiar with Linux should attempt it. As of Red Hat Linux version 4.*x,* the `/RedHat/RPMS` directory contains approximately 170MB of `.rpm` files. RPM compresses these packages and assumes that the packages need an average of 2 to 3MB of hard drive space for every 1MB of RPM package volume. If *package*`.rpm` is 6MB in size, you need between 12 and 18MB of free space to install the package.

Customizing which packages are available for installation is an option when installing the system via FTP, NFS, and hard drive. CD-ROMs (typically) cannot be written to, but you can copy the files to the hard drive and install from there with the customized package list. FTP and NFS installations can only be designed if you have root access to the server(s) on your network or your system administrator is willing to work with you. The following installation situations make custom installation desirable: when obtaining Red Hat Linux via FTP over a low-bandwidth connection or when designing a suite of software to be used by all Red Hat Linux workstations on a network.

To customize the installation, you must obtain the `/base/comps` file which provides you with the list of packages that a full installation would normally include. Then, the packages you actually want to install from `/base/comps` need to be downloaded. The `/base/comps` file must be edited to reflect the packages that you obtained and are going to install.

◇ If you have local RPM packages, you can add them to the `comps` file as well.

The `comps` file. The Red Hat installation program uses the file `/RedHat/base/comps` (the file here is an example from Red Hat Linux version 4.0) to determine what packages are available in the `/RedHat/RPMS` directory for each category to be installed. The file is organized by category, and each category contains a list of packages Red Hat believes

are the minimum required for that section. NOTE: only the *package* part of a package's name (`package-version-build.rpm`) is listed in the file. This means the `comps` file is generally usable from one version of Red Hat to the next. A section in this file has the structure:

```
number category
package
...
end
```

That is a tag to identify the category number, the category, a list of the package names in the category, and the tag "end" to mark the end of the category.

Without exception, everyone needs all of the software packages listed in the **Base** section of the file. The other sections, though, can generally be customized or eliminated to suit a particular need. For example, there are three types of **Networked Stations**: "plain", management, and dial-up. An examination of these sections shows that many of the software packages are listed in all three categories, but some software packages are specific to the category. If you are creating a **Dial-up Networked Station**, then you can safely eliminate the "Plain" and "Management" sections and any software unique to those categories. Conversely, if you only need basic networking capability for networked work stations, the other sections can be eliminated from the file as well as the software unique to those sections. All you need to do is make sure that you have all of the software packages listed in that category. If you have local custom packages (those not provided by Red Hat Software), you should add them to an existing category that is appropriate rather than creating a new category.

Because the list of packages in each category only contains the name of the package (i.e., not the entire `package-name-version-build.rpm`), you can substitute any updates Red Hat has made available in the `updates` directory on:

```
ftp://ftp.redhat.com/pub/redhat/current/updates
```

or one of Red Hat's mirror sites for the original package found in the distribution's original `/RedHat/RPMS` directory. The installation program is relatively version-insensitive. The only warning here is to insure that package dependencies are met. When an RPM package is built, RPM itself tries to determine what packages must be installed for the package to work (the RPM developer also has direct control of this as well—he or she can add dependencies that RPM might not ordinarily detect). This is where a little experimentation

or research may be needed. For example, one way to determine package dependencies (if you have user access to your NFS server on an existing Red Hat Linux box) is to `telnet` or `login` into it (or if you have the CD-ROM, mount it and go to the `RedHat/RPMS` directory) and query the package for its dependencies:

```
[root@happy RPMS] rpm -q -p -R bash-1.14.7-1.i386.rpm

libc.so.5

libtermcap.so.2
```

The "`-q`" puts `rpm` in query mode, the "`-p`" tells `rpm` to query an uninstalled package, and the "`-R`" tells `rpm` to list the target package's dependencies. In this example, we see `libc.so.5` and `libtermcap.so.2` are required. Since `libc` and `termcap` are part of the base of required software (as is `bash`), you must insure that the `libc` and `libtermcap` packages (the dependency packages) are present to be able to install `bash` (the target). As long as you get the entire base system installed, you can boot the system when the installation program completes. You can add additional packages to Red Hat Linux as required even if the installation program reports that a package failed to install because its dependencies were not met.

The table on page 126 describes the categories of software found in `/base/comps` in Red Hat v4.0:

2.4.7 Recommended minimal installation.

It is difficult to determine how much space an installation will require. However, someone installing via FTP should get the **Base** system and the **Dialup Networked Station** and install these. Then, additional software can be obtained and added as the need arises. Of course, if you want to do C programming, you should get the relevant packages and edit the `comps` file appropriately.

◇ If you encounter a package during the installation which requires another package that you don't have available, or you make a mistake in the `comps` file, you can generally finish the installation and have a bootable, working system. You can correct the problem by manually adding the failed packages and their dependencies later. Overall, get the entire **Base**

system and one of the **Networked Station** packages installed, and you can add anything you need or want later.

2.4.8 How much space do you really need?

The table on page 127 gives approximate disk space requirements Red Hat Linux and various subsystems.

2.4.9 Installation.

By now, you should have created the Installation Floppy Kit, prepared your hard drive, and have your installation media ready. The details of the installation follow. You first begin by booting your system and configuring the installation program to install from your selected medium. After this the installation proceeds with the same steps for everyone. You need to begin by booting your computer with the diskette labeled "Boot diskette."

2.4.10 Installation media revisited.

As the boot diskette starts up, the kernel attempts to detect any hardware for which the boot kernel has drivers compiled directly into it. Once booting is complete, a message appears which asks if you have a color screen (if you do, select "`OK`"). Next comes the Red Hat Welcome screen. Choose "`OK`" to continue. The next question asks if you need PCMCIA support. You must answer "`yes`" if you are installing to a laptop, inserting the Supplemental diskette when prompted. Once PCMCIA support is enabled if necessary, you are presented with a screen that asks what type of installation method to use. Follow the instructions in the following sections for the method you chose.

CD-ROM installation. To install from CD-ROM, highlight "`Local CD-ROM`" from the list of installation types. Then click "`OK`". You will be asked if you have a SCSI, IDE/ATAPI, or proprietary CD-ROM. This is where some of the hardware research pays off: if you have 4X or faster CD-ROM drive that was made recently and bundled with a Sound Blaster or other sound card, you most likely have an IDE/ATAPI type drive. This is one of the most confusing issues facing you.

If you choose SCSI, you must know what kind of SCSI card you have and will be presented a list. Scroll down the list until you find your SCSI card. After you select it,

you will be asked if you wish to AUTOPROBE for it or SPECIFY OPTIONS. Most people should choose AUTOPROBE, which causes the program to scan for your SCSI card and enable the SCSI support for your card when found.

After the Installation Program has successfully located the Red Hat CD-ROM, you should read the next section.

Hard drive installation. To install from a hard drive, highlight this option and choose "OK". If you have not already chosen PCMCIA support, you will be prompted to insert the Supplemental diskette.

NFS installation. To install via NFS, highlight this option and choose "OK". You must choose the Ethernet card installed on the target machine so the Installation Program can load the correct driver. Highlight the appropriate card from the list, and then select "OK", allowing the Installation Program to AUTOPROBE for your card.

◇ If your machine locks up, you must press Ctrl-Alt-Delete to reboot the system. Most of the time, when this happens, it is because the probing interferes with a non-Ethernet card. If this happens, try again and choose SPECIFY OPTIONS, and give the data about your card in this form:

```
ether=IRQ,IO_PORT,eth0
```

This instructs the probe to look at the location specified by the values *IRQ* and *IO_PORT* for the Ethernet card. If your Ethernet card is configured for IRQ 11 and IO_PORT 0x300, specify:

```
ether=11,0x300,eth0
```

After the card has been successfully found, you will be prompted for TCP/IP information about your machine and the NFS server with the Linux installation packages. First, you will be asked to provide the target machine's *IP Address*, *Netmask*, *Default Gateway*, and *Primary Name Server*. For example:

```
IP Address:          192.168.181.21
Netmask:             255.255.255.0
Default Gateway:     192.168.181.1
Primary Nameserver:  192.168.181.2
```

After you select OK, you are prompted for the target machine's *Domain name* and *Host name*. For example, if your domain name is `infomagic.com` and host name is `vador`, enter:

```
Domainname:               infomagic.com
Host name:                vador.infomagic.com
Secondary nameserver IP:  Enter if needed
Tertiary nameserver IP:   Enter if needed
```

The last screen prompts you for the NFS server and the exported directory containing the Red Hat distribution. For example, if your NFS server is `redhat.infomagic.com`, enter:

```
NFS Server name:    redhat.infomagic.com
Red Hat Directory:  /pub/mirrors/linux/RedHat
```

If you do not know these values, ask your system administrator. After you enter the values, select `OK` to continue. If the installation program reports an error locating the Red Hat distribution, make sure that you have the correct values filled in above and that your network administrator has given you export permission for the target machine.

FTP installation. FTP installation is similar to the NFS installation described above. You are prompted for the Ethernet card and your machine's TCP/IP information. However, you will be asked for the *FTP site name* and *Red Hat directory* on the Red Hat mirror site, instead of NFS server information. One warning about performing an FTP installation: find the closest and least busy FTP site to your location. See Appendix B for a list of Linux FTP sites.

◇ If your hardware isn't detected, you may need to provide an override for the hardware to be enabled properly. You may also want to check:

`http://www.redhat.com/pub/redhat/updates/`*`version`*`/images`

to see if Red Hat has updated boot diskettes for your hardware.

2.4.11 Walking through the rest of the installation.

1. Next, you are asked if you are installing to a New System or Upgrading Red Hat Linux 2.0 or greater. If upgrading, you will not be offered the chance to partition your hard drive or configure anything with your system except LILO. Select either `INSTALL` or `UPGRADE` to continue.

2. If you are upgrading, you will be asked for the root partition of your existing Red Hat system. Highlight the appropriate partition and press `OK`. If you are installing

for the first time, you need to partition your hard disk with the free space determined above.

3. After you create the necessary Linux Native and Linux Swap partitions, you must initialize and enable the swap partition. You will then be asked to which partition(s) you intend to install Linux If upgrading, select the root partition. You must configure and choose at least one partition, which will be the root partition. Highlight the root partition. Then, unless you are upgrading, you are presented with a table of other available partitions. Choose the appropriate partitions and `EDIT` to indicate which partitions will be used for which directories. If you have more than one partition for the Linux installation, now is the time to designate those as well.

4. Next, a list of software categories to install is presented, followed by a chance to customize which software packages from each category are to be installed. If you have not installed Red Hat or other distributions of Linux before, simply choose the category of software to install and let the setup program install the defaults for each category. If you need a package that wasn't installed originally, you can always install it later. While the software is installing, you will see a progress indicator and you should get a cup or two of coffee. Installation can take thirty minutes to an hour or more, depending on software choices and hardware configuration.

5. After the software installation is done, you will be asked to configure your mouse. A discussion mouse protocols and devices starts on page 40.

6. Next is the X Window System configuration. It is recommend you wait until after you boot your system for the first time to configure X. If something goes wrong with the X configuration, you may need to start the installation procedure from the beginning if the Installation Program isn't able to recover.

7. If you do not have an Ethernet card, *do not* configure your network at this time. If you have a network card and didn't configure it earlier, you should configure it now. Configuration for a dialup network should be done after the installation is complete.

8. Next, you need to configure the system clock. UTC is a good choice if you are on a network and want daylight savings time handled properly. Local Time is okay if the computer is a stand-alone machine.

9. If you do not have a US keyboard, you will need specify the configuration for your keyboard.

10. You are prompted for the `root` system password. Don't forget it. Recovering the password is not a trivial matter. You will need the password to access the system when you first reboot.

11. Finally, you will be asked to configure LILO.

◇ If you have not installed a root partition that begins and ends between cylinder 0-1023, *Do not install LILO.* When you reboot the system for the first time, if LILO does not allow you to boot your system correctly, use the Emergency MS-DOS and Windows 95 boot diskette and, at `A:\>` enter `FDISK /mbr`. This allows your system to boot into an existing MS-DOS or Windows 95 system as it did before LILO was installed. You can then use the Red Hat Boot diskette with the following parameters at the `boot:` prompt to boot your system on the hard drive:

```
boot:  rescue root=/dev/xxxx ro load_ramdisk=0
```

Where *xxxx* is the root partition.

After the installation procedure is complet, you are ready to reboot your system and use Linux.

2.4.12 After installation.

Now that you have installed Linux and booted your system for the first time, there are some useful things to know about using your system

Understanding the LILO prompt. When you power up or reboot the system, you may see the LILO prompt, which you hopefully configured for a 30-second or so delay before it boots the system. When LILO appears on the screen, if you do nothing, the default operating system will boot at the prescribed timeout period. However, from LILO, you can control several aspects of how Linux boots, or tell LILO to boot an alternative operating system. If you wish to override the default behavior of LILO, pressing the `Shift` key at the appearance of LILO will cause a "boot:" prompt to appear. Pressing `Tab` at this prompt will produce a list of available operating systems:

```
LILO boot:
dos linux
boot:
```

This tells us that "`dos`" is the default operating system, which will boot if nothing is typed; to boot Linux, type "`linux`". However, LILO lets you pass parameters to the Linux kernel which override the default behavior. For example, you may have been experimenting with start-up configuration files and did something to prevent the system from coming up properly. If so, you want to boot the system up to the point where it reads the configuration files and no further . The override for this is "`single`":

```
boot:  linux single
```

boots the system in single user mode so you can take corrective action. This is also useful if your system doesn't boot all the way to the `login:` prompt for some reason.

Logging in the first time. Now that you are faced with the `login:` prompt for the first time, you may be wondering how to get into the system. At this point on a newly installed system, there is only one account to log in to—the administrative account, "`root`". This account is used to manage your system and do things like configure the system, add and remove users, software, and so on. To login into the account, enter "`root`" at the `login:` prompt and press `Enter`. You are aked for the password you entered during installation. Enter that password at the `password:` prompt. The system prompt [`root@locahost`] `#` appears after you have successfully negotiated the login. The system prompt tells you two things: you are logged in as `root`, and in this case, your machine is called `localhost`. If you named your machine during the installation process, your host name will appear instead of `localhost`.

2.5 Caldera OpenLinux

This section on Caldera OpenLinux was written by Evan Leibovitch.

This section is intended to be a complement to the Getting Started Guides that Caldera ships with all of its Linux-based products. References to the Getting Started Guide for Caldera Open Linux Base is indicated throughout this section as "the Guide".

2.5.1 Obtaining Caldera OpenLinux.

Unlike most other Linux distributions, Caldera OpenLinux is not available for downloading from the Internet, nor can it be distributed freely, nor passed around. This is because of the commercial packages which are part of COL; while most of the components of COL are under the GNU Public License, the commercial components, such as Looking Glass

and Metro-X, are not. In the list of packages included on the COL media starting on page 196 of the Guide, the commercial packages are noted by an asterisk.

COL is available directly from Caldera, or through a network of Partners around the world who have committed to supporting Caldera products. These Partners can usually provide professional assistance, configuration and training for Caldera users. For a current list of Partners, check the Caldera web site.

2.5.2 Preparing to install Caldera OpenLinux.

Caldera supports the same hardware as any other release based on Linux 2.0 kernels. Appendix A of the Guide lists mosts of the SCSI hosts supported and configuration parameters necessary for many hardware combinations.

Caldera's Guide provides an installation worksheet that assists you in having at hand all the details of your system that you'll need for installation. It is highly recommended you complete this before starting installation; while some parameters, such as setting up your network, are not required for installation, doing it all at one time is usually far easier than having to come back to it. Sometimes this can't be avoided, but do as much at installation time as possible.

2.5.3 Creating boot/modules floppies.

The COL distribution does not come with the floppy disks required for installation. There are two floppies involved; one is used for booting, and the other is a "modules" disk which contains many hardware drivers.

While the Guide recommends that you create the floppies by copying them from the CD-ROM, it is better to get newer versions of the disks from the Caldera web site. The floppy images on older CD-ROMs have errors that cause problems, especially with installations using SCSI disks and large partitions.

To get newer versions of the floppy images, download them from Caldera's FTP site. In directory `pub/col-1.0/updates/Helsinki`, you'll find a bunch of numbered directories. Check out the directories in descending order—that will make sure you get the latest versions.

If you find one of these directories has a subdirectory called `bootdisk`, the contents of that directory are what you want.

You should find two files:

`install-2.0.25-`*XXX*`.img`

`modules-2.0.25-`*XXX*`.img`

The *XXX* is replaced by the version number of the disk images. At the time of writing, the current images are `034` and located in the `001` directory.

After you have these images, transfer them onto two floppies as described for generic installations on page 50, using the MS-DOS program `RAWRITE.EXE` from the Caldera CD-ROM or `dd` from a Linux system.

Caldera's CD-ROM is bootable if your system's BIOS allows it, but use the downloaded floppies if possible. They are newer and will contain bug-fixes that won't be in the CD versions.

2.5.4 Preparing the hard disks.

This procedure is no different than other Linux distributions. You must use `fdisk` on your booted hard disk to allocate at least two Linux partitions, one for the swap area and one for the root file system. If you are planning to make your system dual-boot COL with another operating system, like Microsoft Windows, MS-DOS, or OS/2, it's usually preferable to install COL last. The Linux `fdisk` programs recognizes "foreign" OS types better than the disk partitioning tools of most other operating systems.

To run the Linux `fdisk`, you must start your system with the boot (and maybe the modules) floppy described above. You must tell COL what kind of disk and disk controller you have. You can't even get as far as entering `fdisk` if Linux doesn't recognize your hard disk!

To do this, follow the bootup instructions in the Guide, from step 2 on pages 33–36. Don't bother going through the installation or detection of CDROMs or network cards at this time; all that matters at this point is that Linux "sees" the boot hard disk so you can partition it with `fdisk`. A brief description of the use of the Linux `fdisk` is provided on page 28 of the Guide.

Remember that when running `fdisk`, you need to set up both your root file system as Linux Native (type 83) and your Swap space (type 82) as new partitions. A brief discussion of how much swap space to allocate is offered on page 10 of the Guide.

◇ As soon as you have allocated the partitions and written the partition table information to make it permanent, you must reboot.

2.6 Slackware

This section on Linux Slackware was written by Sean Dreilinger.

2.6.1 Slackware is not for you. (Or maybe it is.)

Welcome to the Slackware distribution of Linux! This section aims to help the new Linux user or administrator evaluate Slackware, plan a Slackware system, and install Slackware Linux.

Whether or not to choose Slackware as the flavor of Linux you will use is a serious consideration. It may seem like a trivial decision now, but Linux boxes have a way of taking on more and more responsibility in organizational computing environments. Plenty of Linux *experiments* have evolved in their first year to become mission-critical machines serving many more users and purposes than originally intended. Slackware is one of the most widely used distributions of Linux. When it comes to finding the newest, easiest, or most carefully planned distribution of Linux, Slackware may be "none of the above". Some background on the life and times of Slackware put things into perspective.

2.6.2 A quick history.

In 1993, Soft Landing System created one of the first organized distributions of Linux. Although it was a great start, the SLS distribution had many shortcomings (it didn't exactly work, for starters). Slackware, a godsend from Patrick Volkerding, solved most of these issues, was mirrored via FTP and pressed onto CD-ROMs worldwide, and quickly became the most widely used flavor of Linux. For a while, Slackware was the only full featured Linux "solution." Other Linux distribution maintainers, both commercial and nonprofit, have gradually developed distributions that are also well worth your consideration.

By January 1994, Slackware had achieved such widespread use that it earned a popular notoriety normally reserved for rock stars and cult leaders. Gossip spread through the Usenet suggesting that the entire Slackware project was the work of witches and devil-worshippers!

"Linux, the free OS....except for your SOUL! MOUHAHAHAHA!"

```
From:  cajho@uno.edu
Date:  7 Jan 1994 15:48:07 GMT


Jokes alluding to RFC 666, demonic daemons, and
speculation that Pat Volkerding was actually L. Ron
Hubbard in disguise were rampant in the threads that
followed.  The whole amusing incident probably helped
Slackware gain some market share:

I LOVE THIS!!
```

```
I was browsing here to figure which version of Linux to
install, but after this, I think that I hve no choice but
to install Slackware now.

From:  widsith@phantom.com (David Devejian)
Date:  10 Jan 1994 04:57:41 GMT


All folklore and kidding aside, Slackware is a wise and
powerful choice for your adventures in Linux, whether you
are a hobbyist, student, hacker, or
system-administrator-in-the-making.
```

2.6.3 Why, then?

If you are a system administrator, you may already be dealing with one or more key servers running Slackware. Unless you have time to experiment at work, sticking to the tried-and-true distribution may be the easiest way to go. If you expect to get help from UNIX literate friends and colleagues, you had better make sure they're running something compatible—odds are they're running Slackware. Its shortcomings are widely acknowledged, for the most part discovered, documented, and patched whenever possible. You can put together a Slackware box, close the known security holes, and install some complementary tools from the other Linux distributions to create an excellent UNIX server or desktop workstation, all in about half a day.

Have a look also at the Buyer's Guide published in the *Linux Journal*, which gives a thorough comparison and evaluation of each major distribution. For a straightforward listing of Linux flavors, have a look at the Linux Distribution HOWTO (see Appendix A).

2.6.4 Upgrade? Think twice!

```
24-Aug-95 NOTE: Trying to upgrade to ELF Slackware from
a.out Slackware will undoubtedly cause you all kinds of
problems.  Don't do it.

Patrick Volkerding
```

One thing we don't hear too often with Slackware is the U-word. Slackware's setup program is designed to put a fresh operating system onto empty hard disks or empty disk partitions. Installing on top of a previous Slackware installation can erase your custom applications and cause compatibility problems between updated applications and older files

on the same system. When Slackware was first put together, everyone was a first-time Linux user, and the system was always experimental—reinstalling the entire operating system and applications was the norm in a developmental system. Today, many institutions and businesses run mission-critical applications on Slackware Linux. In such environment, a simple reboot is a planned activity and taking down the system and overwriting all the user files or custom applications is absolutely unacceptable.

Teaching you how to finagle a Slackware upgrade is beyond the scope of this chapter, but it is workable if you are an experienced UNIX administrator and you've taken precautions to preserve your local modifications and user files. There is an Internet resource that claims to analyze your distribution and bring it up to date across the Internet. you might want to have a look at this URL if you're facing an upgrade situation:

```
ftp://ftp.wsc.com/pub/freeware/linux/update.linux/
```

Or read, weep, and learn from the upgrade expertise of Greg Louis in his mini HOWTO document: *Upgrading Your Linux Distribution* available where finer LDP publications are mirrored:

```
http://sunsite.unc.edu/LDP/
```

2.6.5 Select an installation method.

Slackware can be installed from a variety of media and network sources to fit your needs and budget. Every installation method requires you to have at least three floppy diskettes available to get started.

CD-ROM. Installation from CD-ROM is fast, popular, and convenient. Although someone has to break down and pay for the initial purchase of a CD-ROM, sharing CD's is *encouraged*. Because Linux and the Slackware distribution are copylefted, you may make as many copies as you like. CD-ROM installation is also a bit better practice in terms of netiquette, since you're not hogging bandwidth for an all-day FTP transfer. Finally, you may be grateful for the extra utilities and documentation that accompany the CD-ROM, especially if you run into installation hassles or need to add components in the future.

Party! If you're a hobbyist (or want to watch a few dozen Slackware installs before taking on the task at work), see if there is a LUG (Linux User Group) in your area that sponsors install parties. Imagine a roomful of generous and knowledgeable hackers uniting to share CD-ROMs and expertise with other enthusiasts.

FTP. Once you transfer Slackware from the closest possible FTP mirror, you'll still need to put the Slackware 'disk sets' onto installation media such as a hard drive partition or laboriously copy them onto 50-odd floppy diskettes.

NFS. In a networked environment, it is possible to install Slackware on a shared file system and allow everyone on the Local net to attach to this shared location and install. If you have the technical know-how or a geeked out system administrator who is Linux-literate, this is a great way to go. The initial distribution of Slackware can be added to the network via CD-ROM, FTP, Loading floppies, tape, or even via a remote NFS share across the Internet! For details on such a remote share, see these URLs:

```
http://sunsite.doc.ic.ac.uk/sunsite/access/nfs.html
ftp://ftp.cdrom.com/pub/linux/slackware/MIRRORS.TXT
http://www.cs.us.es/archive/nfs.html
```

Floppy. It's time consuming, but it works—you can create the pile of floppies needed to install Slackware and then feed them into your box one-by-one when prompted. Slackware "disk sets" are actually designed and arranged to fit floppy diskettes. If you happen to have a huge stack of recycled, high-density floppy diskettes at your disposal, this can be the most economical way to go.

Hard disk. This is the way to do it if you've transferred the Slackware distribution across the Internet via FTP—you'll escape the floppy trap by merely creating boot, root, and rescue diskettes. It requires you to have an extra disk or disk partition with extra space to hold the Slackware files during installation (you can erase them afterwards). Installation from the hard drive is also a workaround if you bought the CD but your CD-ROM drive is not supported by any of the Linux kernels that come with the Slackware CD. You can use your present operating system to transfer the Slackware files onto spare hard disk space, then boot into the Slackware installation.

Tape. Still experimental as of this writing, tape offers a great compromise of speed and economy when installing Slackware—worth considering if a friend with compatible tape drive can dupe a CD or FTP archive for you. Get the latest details from the Tape section of the `INSTALL.TXT` file that accompanies your Slackware distribution.

2.6.6 Boot disks: always a good thing.

Even if you're gifted with a direct T-3 Internet connection that allows you to suck up a new distribution of Slackware right off the 'Net, you'll be wise to start by building the two Slackware setup disks (boot and root) before proceeding. In the event of an unfortunate accident (power outage, feline friends traversing the keyboard, or even human error), these two little disks may be able to revive your system or at least rescue your personal files.

2.6.7 Slackware setup worksheet.

After the files are all copied, Slackware can go on to do most of the system and network configuration, if you're ready. To help you plan your decisions, this section consists of a worksheet derived from the text-based Slackware setup program. You can use this worksheet to record answers in advance (while your computer is still working!), so you'll be ready with the necessary details-partitions, IP addresses, modem and mouse IRQs, host and domain names, and others that you're required to provide during setup.

1. **Keyboard:** Slackware `setup` will want to know if you need to remap your keyboard to something other than a standard USA 101 key layout?

   yes or no

2. **Swap Configuration:** Do you have one or more partitions prepared as type 82 (Linux Swap)?

   yes or no

   Do you want `setup` to use `mkswap` on your swap partitions? Most likely "yes", unless you have less than 4MB of RAM and have already done this to help `setup` work better.

   yes or no

3. **Prepare Main Linux Partition:** `setup` will list any partitions marked as type 83 (Linux Native) and ask which one to use for the root (/) of the Linux file system. Use a format like `/dev/hda3` or whatever the device name is.

   partition name

◇ Last chance to back out! When using the install from scratch option, you must install to a blank partition. If you have not already formatted it manually, then you must format it when prompted. Enter "`I`" to install from scratch, or "`a`" to add software to your existing system.

[i]nstall or [a]dd

(Re)format the main Linux partition. Would you like to format this partition?

[y]es, [n]o, or [c]heck sectors, too

`ext2fs` defaults to one inode per 4096 bytes of drive space. If you're going to have many small files on your drive, you may need more inodes (one is used for each file entry). You can change the density to one inode per 2048 bytes, or even per 1024 bytes. Enter `2048` or `1024`, or just hit `Enter` to accept the default of 4096.

4096 (default). 2048, or 1024

4. **Prepare Additional Linux Partitions:** You can mount some other partitions for `/usr` or `/usr/X11` or whatever (`/tmp`—you name it). Would you like to use some of the other Linux partitions to mount some of your directories?

[y]es or [n]o

These are your Linux partitions (*partition list displayed*). These partitions are already in use (*partition list displayed*). Enter the partition you would like to use, or type `q` to quit adding new partitions. Use a format such as: `/dev/hda3` or whatever the device name is.

Partition name or [q]uit

Would you like to format this partition?

[y]es, [n]o, or [c]heck sections, too

Now this new partition must be mounted somewhere in your new directory tree. For example, if you want to put it under `/usr/X11R6`, then respond: `/usr/X11R6` Where would you like to mount this new partition?

Mount point

Would you like to mount some more additional partitions?

[y]es or [n]o

5. **DOS and OS/2 Partition Setup:** The following DOS FAT or OS/2 HPFS partitions were found: (*partition list displayed*). Would you like to set up some of these partitions to be visible from Linux?

   [y]es or [n]o

   Please enter the partition you would like to access from Linux, or type `q` to quit adding new partitions. Use a format such as: `/dev/hda3` or whatever the device name is.

   Partition name or [q]uit

   Now this new partition must be mounted somewhere in your directory tree. Please enter the directory under which you would like to put it. for instance, you might want to reply `/dosc`, `/dosd`, or something like that. Where would you like to mount this partition?

   Mount point

6. **Source Media Selection:**

```
(a) Install from a hard drive partition.
(b) Install from floppy disks.
(c) Install via NFS.
(d) Install from a pre-mounted directory.
(e) Install from CD-ROM.
```

1, 2, 3, 4, or 5

7. **Install from a hard drive partition:** To install directly from the hard disk, you must have a partition with a directory containing the Slackware distribution such that each disk other than the boot disk is contained in a subdirectory. For example, if the distribution is in `/stuff/slack`, then you need to have directories named `/stuff/slack/a1`, `/stuff/slack/a2`, and so on, each containing the files that would be on that disk. You may install from DOS, HPFS, or Linux partitions. Enter the partition where the Slackware sources can be found, or [p] to see a partition list.

   Partition name or [p]artition list

   What directory on this partition can the Slackware sources be found. In the example above, this would be: `/stuff/slack`. What directory are the Slackware sources in?

   Directory name

   What type of file system does your Slackware source partition contain?

   ```
   (a) FAT (MS-DOS, DR-DOS, OS/2)
   (b) Linux Second Extended File System
   (c) Linux Xiafs
   (d) Linux MINIX
   (e) OS/2 HPFS
   ```

   1, 2, 3, 4, or 5

8. **Install from a pre-mounted directory:** Okay, we will install from a directory that is currently mounted. This can be mounted normally or through NFS. You need to specify the name of the directory that contains the subdirectories for each source disk. Which directory would you like to install from?

   Directory name

9. **Install from floppy disks:** The base Slackware series (A) can be installed from 1.2M or 1.44M media. Most of the other disks will not fit on 1.2M media, but can be downloaded to your hard drive and installed from there later. Which drive would you like to install from (1/2/3/4)?

```
/dev/fd0u1440 (1.44M drive a:)
/dev/fd1u1440 (1.44M drive b:)
/dev/fd0h1200 (1.2M drive a:)
/dev/fd1h1200 (1.2M drive b:)
```

1, 2, 3, or 4

10. **Install via NFS:** You're running off the hard drive file system. Is this machine currently running on the network you plan to install from? If so, we won't try to reconfigure your ethernet card. Are you up and running on the network?

    [y]es or [n]o

    You will need to enter the IP address you wish to assign to this machine. Example: 111.112.113.114. What is your IP address?

    IP address

    Now we need to know your netmask. Typically this will be 255.255.255.0. What is your netmask?

    IP address

    Do you have a gateway (y/n)?

    [y]es or [n]o

    What is your gateway address?

    IP address

    Good! We're all set on the local end, but now we need to know where to find the software packages to install. First, we need the IP address of the machine where the Slackware sources are stored. Since you're already running on the network, you should be able to use the hostname instead of an IP address if you wish. What is the IP address of your NFS server?

IP address

There must be a directory on the server with the Slackware sources for each disk in subdirectories beneath it. `setup` needs to know the name of the directory on your server that contains the disk subdirectories. For example, if your A3 disk is found at `/slackware/a3`, then you would respond: `/slackware`. What is the Slackware source directory?

Directory name

11. **Install from CD-ROM:** What type of CD-ROM drive do you have?

```
(a) Works with most ATAPI/IDE CD drives (/dev/hd*)
(b) SCSI (/dev/scd0 or /dev/scd1)
(c) Sony CDU31A/CDU33A (/dev/sonycd)
(d) Sony 531/535 (/dev/cdu535)
(e) Mitsumi, proprietary interface---not IDE
(/dev/mcd)
(f) New Mitsumi, also not IDE (/dev/mcdx0)
(g) Sound Blaster Pro/Panasonic (/dev/sbpcd)
(h) Aztech/Orchid/Okano/Wearnes (/dev/aztcd)
(i) Phillips and some ProAudioSpectrum16
(/dev/cm206cd)
(j) Goldstar R420 (/dev/gscd)
(k) Optics Storage 8000 (/dev/optcd)
(l) Sanyo CDR-H94 + ISP16 soundcard (/dev/sjcd)
(m) Try to scan for your CD drive
```

1, 2, 3, 4, 5, 6, 7 8, 9, 10, 11, 12, or 13

IDE CD-ROM: Enter the device name that represents your IDE CD-ROM drive. This will probably be one of these (in the order of most to least likely): `/dev/hdb /dev/hdc /dev/hdd /dev/hde /dev/hdf /dev/hdg /dev/hdh /dev/hda`

Device name

SCSI CD-ROM: Which SCSI CD-ROM are you using? If you're not sure, select `/dev/scd0`.

1. /dev/scd0
2. /dev/scd1

installation method: With the Slackware CD, you can run most of the system from the CD if you're short of drive space or if you just want to test Linux without going through a complete installation. Which type of installation do you want (slakware or slaktest)?

slakware Normal installation to hard drive

slaktest Link `/usr->/cdrom/live/usr` to run mostly from CD-ROM

slakware or slaktext

12. **Series Selection:** Identify which Packages you plan to install. You may specify any combination of disk sets at the prompt which follows. For example, to install the base system, the base X Window System, and the Tcl toolkit, you would enter: `a x tcl`. Which disk sets do you want to install?

```
A Base Linux system
AP Various applications that do not need X
D Program Development (C, C++, Kernel source, Lisp,
Perl, etc.)
E GNU Emacs
F FAQ lists
K Linux kernel source
N Networking (TCP/IP, UUCP, Mail)
Q Extra kernels with special drivers (needed for
non-SCSI CD)
T TeX
TCL Tcl/Tk/TclX, Tcl language, and Tk toolkit for
developing X apps
X Xfree86 Base X Window System
XAP X Window Applications
XD Xfree86 X11 server development system
```

```
XV Xview (OpenLook virtual Window Manager, apps)
Y Games (that do not require X)
```

Any combination of a ap d e f k n q t tcl x xap xd xv y and other disk sets offered, separated by spaces

13. **Software Installation:** Next, software packages are going to be transferred on to your hard drive. If this is your first time installing Linux, you should probably use PROMPT mode. This will follow a defaults file on the first disk of each series you install that will ensure that required packages are installed automatically. You will be prompted for the installation of other packages. If you don't use PROMPT mode, the install program will just go ahead and install everything from the disk sets you have selected. Do you want to use PROMPT mode (y/n)?

    [y]es or [n]o

    These defaults are user definable—you may set any package to be added or skipped automatically by editing your choices into a file called `TAGFILE` that will be found on the first disk of each series. There will also be a copy of the original tagfile called `TAGFILE.ORG` available in case you want to restore the default settings. The tagfile contains all the instructions needed to completely automate your installation. Would you like to use a special tagfile extension? You can specify an extension consisting of a "." followed by any combination of 3 characters other than `tgz`. For instance, I specify ".pat", and then whenever any tagfiles called "tagfile.pat" are found during the installation they are used instead of the default "tagfile" files. If the install program does not find tagfiles with the custom extension, it will use the default tagfiles. Enter your custom tagfile extension (including the leading "."), or just press `Enter` to continue without a custom extension.

    Tagfile extension `Enter`

14. **Extra Configuration:** If you wish, you may now go through the options to reconfigure your hardware, make a bootdisk, and install LILO. If you've installed a new kernel image, you should go through these steps again. Otherwise, it's up to you.

    [y]es or [n]o

15. **Boot Disk Creation:** It is recommended that you make a boot disk. Would you like to do this?

    [y]*es or* [n]*o*

    Now put a formatted floppy in your boot drive. This will be made into your Linux boot disk. Use this to boot Linux until LILO has been configured to boot from the hard drive. Any data on the target disk will be destroyed. Insert the disk and press `Return`, or `s` if you want to skip this step.

    `Enter` *or* [s]*kip*

16. **Modem Setup:** A link in `/dev` will be created from your callout device (`cua0`, `cua1`, `cua2` , `cua3`) to `/dev/modem`. You can change this link later if you put your modem on a different port. Would you like to set up your modem?

    [y]*es or* [n]*o*

    These are the standard serial I/O devices, Which device is your modem attached to (0, 1, 2, 3)?

    ```
    0 /dev/ttyS0 (or COM1:  under DOS)
    1 /dev/ttyS1 (or COM2:  under DOS)
    2 /dev/ttyS2 (or COM3:  under DOS)
    3 /dev/ttyS3 (or COM4:  under DOS)
    ```

    0, 1, 2, or 3

17. **Mouse Setup:** A link will be created in `/dev` from your mouse device to `/dev/mouse`. You can change this link later if you switch to a different type of mouse. Would you like to set up your mouse?

    [y]*es or* [n]*o*

    These types are supported. Which type of mouse do you have (1, 2, 3, 4, 5, 6, 7)?

```
(a) Microsoft compatible serial mouse
(b) QuickPort or PS/2 style mouse (Auxiliary port)
(c) Logitech Bus Mouse
(d) ATI XL Bus Mouse
(e) Microsoft Bus Mouse
(f) Mouse Systems serial mouse
(g) Logitech (MouseMan) serial mouse
```

1, 2, 3, 4, 5, 6, or 7

These are the standard serial I/O devices. Which device is your mouse attached to (0, 1, 2, 3)?

```
0 /dev/ttyS0 (or COM1:  under DOS)
1 /dev/ttyS1 (or COM2:  under DOS)
2 /dev/ttyS2 (or COM3:  under DOS)
3 /dev/ttyS3 (or COM4:  under DOS)
```

0, 1, 2, or 3

18. **Network Configuration:** Now we will attempt to configure your mail and TCP/IP. This process probably won't work on all possible network configurations, but should give you a good start. You will be able to reconfigure your system at any time by typing `netconfig`. First, we'll need the name you'd like to give your host. Only the base host name is needed right now (not the domain). Enter the host name.

Host name

Now, we need the domain name. Do not supply a leading "`.`" Enter the domain name.

Domain name

If you only plan to use TCP/IP through loopback, then your IP address will be 127.0.0.1, and we can skip a lot of the following questions. Do you plan to *only* use loopback?

[y]es or [n]o

Enter your IP address for the local machine. Example: 111.112.113.114. Enter the IP address for this machine (aaa.bbb.ccc.ddd).

IP address

Enter your gateway address, such as 111.112.113.1. If you don't have a gateway, you can edit `/etc/rc.d/rc.inet1` later,or you can probably get away with entering your own IP address here. Enter the gateway address (aaa.bbb.ccc.ddd).

IP address

Enter your netmask. This will generally look something like this: 255.255.255.0. Enter the netmask (aaa.bbb.ccc.ddd).

IP address

Will you be accessing a name server?

[y]es or [n]o

Please give the IP address of the name server to use. You can add more Domain Name Servers by editing `/etc/resolv.conf`. Name server for your domain (aaa.bbb.ccc.ddd)?

IP address

You may now reboot your computer by pressing `Ctrl`-`Alt`-`Delete`. If you installed LILO, remove the boot disk from your computer before rebooting. Don't forget to create your `/etc/fstab` if you don't have one (see page 196)!

2.6.8 Making Slackware happen.

If you've taken the time to plot and plan as recommended in the preceding sections, then the actual installation is a piece of cake. There isn't much writing needed to explain the actual process of loading Slackware on your computer(s). Follow the steps to build boot and root diskettes, then answer a long series of questions asked by the menu driven Slackware installation program. If you've completed the Slackware Installation Worksheet, these questions will be familiar and everything will run smoothly.

2.6.9 Build some boot disks.

Choose your kernel! When installing Slackware Linux, you must create a boot diskette with a Linux kernel that is specially prepared to recognize your system hardware. For example, to install Slackware from an IDE CD-ROM drive onto a SCSI hard drive, the kernel that you put onto the boot diskette will need to have drivers for your SCSI card and your IDE CD-ROM drive.

The kernels are stored as compressed *binary image* files that you can access from most any operating system to create a Slackware Boot diskette. On the Slackware FTP site, CD-ROM, or NFS mount, you'll find a subdirectory called `bootdsks.144`, containing 1.44 MB kernel images for creating boot disks on 1.44MB high density 3.5" floppy diskettes. If you're working from a 5.25" floppy diskette drive, look in a directory called `bootdsks.12` for kernel images that will fit the smaller diskette format.

The table on page 128 provides a quick reference of the kernel images available as we went to press. Information and up-to-date boot disk image information is available from this URL:

```
ftp://ftp.cdrom.com/pub/linux/slackware/bootdsks.144/README.TXT
```

2.6.10 Boot into action.

Here's the big anticlimax. After all of this planning, preparation, and partitioning, you're in the home stretch. Make sure that the boot floppy is in the diskette drive, and restart your computer. Now is a good time to go get some coffee (or whatever you like to keep you company) and return to the machine ready to play the part of a button-pushing drone, answering yes-no questions for an hour or so.

Log in as root (no password) and type `setup` or `setup.tty`.

2.6.11 The Slackware `setup` program.

Slackware comes with two versions of an excellent `setup` program. One is a colorful, dialog-based, menu-driven version. An alternative, `setup.tty`, is a text-only version that you may actually prefer, because detailed diagnostics and error messages stay on the screen and are not covered up by the next dialog box. If you're attempting a Slackware installation on sketchy hardware, I strongly recommend the less colorful `setup.tty` routine. If you

don't know much about UNIX and would feel more comfortable with an attractive, "clean" interface to the same process, then by all means go for the beautiful setup.

```
============== Slackware96 Linux Setup (version HD-3.1.0) ==============

  Welcome to Slackware Linux Setup.

  Hint: If you have trouble using the arrow keys on your keyboard,
  you can use '+', '-', and TAB instead. Which option would you like?
  ========================================================================
         HELP       Read the Slackware Setup HELP file
         KEYMAP     Remap your keyboard
         MAKE TAGS  Tagfile customization program
         TARGET     Select target directory [now: / ]
         SOURCE     Select source media
         DISK SETS  Decide which disk sets you wish to install
         INSTALL    Install selected disk sets
         CONFIGURE  Reconfigure your Linux system
         PKGTOOL    Install or remove packages with Pkgtool
         EXIT       Exit Slackware Linux Setup
  ========================================================================
================================================================================
                          <  OK  >      <Cancel>
================================================================================
```

Transferring Slackware onto your system from here should involve little more than selecting what you want from the menus. By filling out Section 3 of the worksheet in advance, you should be able progress quickly through each menu in order, until you reach the INSTALL option, at which point things may s l o w down: you are advised to select the PROMPT feature and *read* about each software package, deciding whether or not you'd like it to end up on your Slackware system. The last part of a regular setup is the CONFIGURE section on the setup menu, and the questions you must answer bear a striking resemblance to the second half of the Section 3 worksheet.

2.6.12 Is that all?

Definitely not! At this point, you either have some annoying obstacle that is preventing the setup from completing, or more likely, you're looking at the `root` prompt

```
darkstar~#
```

and wondering "What Next?"

Well, if you're plagued by problems, you'll want to proceed directly to the next section on troubleshooting. If things appear to be in working order, you've still got some details to attend to. It's sort of like purchasing a new automobile—after you select and pay for a car, there are still some things that you need before you can drive it with confidence—insurance, a steering wheel club, and perhaps some luxuries that make the driving experience closer to Fahrvergnügen than FAQ!

2.6.13 Troubleshooting difficult deliveries.

Not every Slackware installation is born on cue to expecting system administrators. I've pulled a few all-nighters, sitting down after work one evening to upgrade a Slackware box and still there struggling to get the damn thing back online at dawn, before people start bitching about their missing mail and news. This section will look at a few common Slackware setup problems, solutions, and where to look for additional assistance.

Slackware installation FAQs. Patrick Volkerding, the father of Slackware, has dealt with many questions of new users by listening, answering, and anticipating repeat queries. To catch the new Slackware users before they ask the same question for the 5,000th time, Patrick has kindly created documentation and included it with the Slackware distribution. Three files that you may find very helpful in answering your initial questions are `FAQ.TXT`, `INSTALL.TXT`, and `BOOTING.TXT`.

Web Support For Slackware. The easiest way to access finding Linux documents in general is the Linux Documentation Project Home Page. See page 30 for a description of the LDP Home Page.

At this time, the Slackware-specific help you'll find on the Internet tends to be highly customized—like how to NFS-mount the distribution on computers within a certain university or how to wire your dorm room into a particular residential WAN using Slackware.

Usenet Groups For Slackware. The `comp.os.linux.*` hierarchy of Usenet is a treasure trove of Linux information, not necessarily Slackware-specific. At present, 11 separate Linux forums handle a high volume of discussion in this hierarchy, which is described on page 31.

Mailing lists for Slackware. At this time, there are no electronic mail discussions devoted to Slackware per se. You can participate in some excellent Linux-related talk via e-mail, try `http://www.linux.org`, and ask in the Usenet newsgroups for a few good subscription lists.

There is a general Linux mailing list server, `majordomo@vger.rutgers.edu`. See page 33 for a description of how to subscribe to mailing lists via this server.

You get what you pay for (commercial support). Commercial support for Linux is available from some of the CD-ROM vendors and a long list of Linux Consultants, who can be contacted through the Linux Commercial and Consultants HOWTO documents:

```
http://sunsite.unc.edu/LDP/HOWTO/Consultants-HOWTO.html
http://sunsite.unc.edu/LDP/HOWTO/Commercial-HOWTO.html
```

2.6.14 Basking in the afterglow.

Don't rest on your laurels quite yet, especially if your Slackware machine is a shared computer or lives in a networked environment. Grooming a computer for community and network use is a bit more demanding than just running the setup program and then forgetting about it. We'll leave you with a few pointers to securing and sharing your new Slackware system.

2.6.15 Consider reinstalling!

I know you just sat through what may have been a long and perplexing installation session. But before you move into the house you just built, consider tearing it down and starting over again. Friedrich Nietzsche had a quote:

> A man learns what he needs to know about building his house only after he's finished.

If, in the process of installing the system, you had some thoughts about how you might do it differently, now is the time. If your Slackware Linux box will be a multi-user machine or a network server, there may never be such a convenient opportunity to re-install or reconfigure the system in radical ways.

2.6.16 Secure the system.

Get off the LAN at once. Out of the box, Slackware is an insecure system. Although Patrick Volkerding does his best to create a secure distribution, a few inevitable holes become known, and patches or workarounds are made available in the system administration (and cracker) communities. If you installed Slackware from a network source like a NFS-mounted drive, you should temporarily disconnect your box from the LAN after a successful installation, while you plug a few holes.

Give `root` a password. By default, a new Slackware box will not require a password for the root user. When you're comfortable that your new Slackware system is stable (after a few hours, not days or weeks), add a password to protect the root account. Login as root and type:

```
# passwd root
```

Give yourself an account. On large, shared systems, the super-user `root` account is not used as a working `login` account by any individual. If you're interested in system administration or are running a networked machine, this is a good precedent to follow. Use the `/sbin/adduser` program and make yourself a `login` account, rather than working out of the `root` login. I always smile when I see students and hobbyists posting proudly to the Usenet as `root@mymachine.mydomain`. Be humble and safe: create another `login` account for your daily work and use `su` (rather than `login`) to enter the `root` account sparingly.

Read Chapter 4 for a discussion of what you should do with the `root` account (or shouldn't).

Deny `root` logins. Not only is it uncommon to *work* as the root user, it is *not considered secure to* `login` *as root across the network.* Administrative users usually connect to a UNIX box as their regular, user-name `login`, then `su` to `root` as needed. To prevent crackers, hackers, and ignorant users from logging in directly as `root`, edit the file `/etc/securetty` and comment out (prepend a pound (#) sign before) all but the local terminals:

```
console

tty1
```

```
tty2
# ttyS0
# ttyS1
```

After this fix, users who attempt to login in as `root` across the network will be denied:

```
Linux 2.0.29 (durak.interactivate.com) (ttyp4)

durak login:  root
root login refused on this terminal.
durak login:
```

Apply the simple fixes. Slackware installs itself with some very real security problems. Rather than master UNIX security and sleuth out these vulnerabilities yourself, you can jump start the hole-patching process by visiting a Web resource maintained for just this purpose, called *Slackware SimpleFixes*:

```
http://cesdis.gsfc.nasa.gov/linux-web/simplefixes/simplefixes.html
```

Check for patches on `ftp.cdrom.com` As an actively maintained Linux distribution, Slackware updates and patches are available from:

```
ftp://ftp.cdrom.com/pub/linux/slackware/patches/
```

Stay current. You might like to subscribe to one or more electronic mailing lists that alert users to issues in Linux administration, such as:

```
linux-alert-request@tarsier.cv.nrao.edu
linux-security-request@redhat.com
```

2.6.16.1 Back up.

Like how things are running? Save it for a rainy day by backing up. Amanda (the Advanced Maryland Automatic Network Disk Archiver) is one of several backup options for Linux installations. You can learn more about Amanda from:

```
http://www.cs.umd.edu/projects/amanda/index.html
```

2.7 S.u.S.E.

This section on S.u.S.E. Linux was written by Larry Ayers.

The SuSE distribution began a few years ago as an adaptation of Slackware. Patrick Volkerding of Slackware helped the SuSE developers at first, but before too long, the distribution began to assume an identity of its own. Several new features intended to aid the first-time user increase the probability an installation won't need to be immediately redone. Given the cross-pollination endemic in the free software world, I wouldn't be surprised to learn some of these features have shown up in newer Slackware releases.

2.7.1 Beginning the installation.

When booting your machine from the single installation disk, you are really booting a miniature Linux system designed for this purpose. A colored screen appears, ready to ask a series of questions which with any luck will guide you through the process. YAST (Yet Another Set-up Tool) shows its Slackware ancestry inasmuch as it uses the `dialog` program. This tool enables shell scripts to present dialog boxes, radio buttons, and check lists which allow a user to make choices and direct the course of an installation.

While no distribution can guarantee a painless installation, the developers at S.u.S.E. GmbH have managed to anticipate several problems that new Linux users are liable to have. One of the more frustrating problems is finding that your CD-ROM drive isn't recognized. Copying the packages needed to get started to a hard drive and installing them from there is a solution, but it's awkward and time-consuming. Rather than provide a selection of several disk images, one of which probably has the CD-ROM drive support you need, the single S.u.S.E. boot disk contains a small, basic kernel with all drivers available–if needed–in the form of modules. The kernel daemon is a background process which ensures the relevant module will be loaded if a modular function is needed. This helps to eliminate one stumbling block. Another common trap is underestimating the disk space which you need. This forces the installation to abort itself due to lack of room. When this happens, the crucial final steps (like LILO installation) haven't yet been reached, and starting over is usually necessary. Script-based installations are necessarily sequential in nature; you may know that skipping one step won't hurt anything, but it's hard to anticipate every eventuality in a shell script, and if things go awry the script usually aborts.

During the S.u.S.E. installation, a running tally of partition space remaining is displayed on the YAST screen; while selecting packages, you can try various combinations while keeping in mind how much free disk space you would prefer to remain free. Partitioning and formatting disks, as well as creation and activation of a swap partition, are processes

that aren't much different than in other distributions. They all use the same underlying tools; the procedure has become more or less standardized.

Dependencies. The use of **dependencies**, which consist of information included in a software package concerning what other packages are necessary for it to run, has spread rapidly among Linux distributions. Unfortunately no universal format for dependencies has arisen. Each distribution uses a different format. Redhat's RPM format, used in several distributions, is powerful and effective, but it has a few drawbacks. It works best on an all-RPM system, as the dependency checking done by the RPM program only knows about RPM packages. S.u.S.E. 5.1 uses srpm-format. The dependencies are only checked if a package is installed from within the YAST program, allowing the option (for a skilled user) of unarchiving a package in another location, then checking out the files and configuration before final installation. Dependencies are most useful during the initial setup and while becoming familiar with a new installation. Once you've used the system for a while, you'll have an idea of what libraries and programs are available. Most software packages for Linux also contain information as to what needs to be present on a system in order for the package to function. It is wise to read through the entire `rc.config` file before running `SuSEconfig` and committing any changes you may have made. Some of the default actions the script will take you may prefer to handle yourself, but they are easily disabled by editing the file.

Users familiar with the Slackware layout of initialization files will need to make some adjustments; the files usually found in `/etc/rc.d` are instead in `/sbin/init.d`.

2.7.2 S.u.S.E Post-installation.

YAST is also intended to be used after installation for routine system maintenance. The multiplicity of resource files necessary for Linux to boot and run can be bewildering to beginners. YAST offers a menu-driven interface to these files, including the `sendmail` configuration file, the `cron` (scheduling) files, initialization scripts, and various networking files. The changes made within the YAST session are written to a single file in the `/etc` directory, `rc.config`, which can also be edited directly. These changes are then written to the various "real" configuration files by a script called `SuSEconfig`. This script is automatically run by YAST at the end of a YAST session; if `/etc/rc.config` is edited directly, `SuSEconfig` must be started manually. This sounds like a complicated procedure, but it's much easier than tracking down the individual files, learning the correct syntax needed to edit them, and making them do what you want.

After you have S.u.S.E. Linux up and running, it's a good idea to install the kernel source (available on the CD-ROM, it's an optional package which can be installed during initial set-up). S.u.S.E. installs a generic kernel, and you probably need only a few of of the accompanying modules. This is an excellent opportunity to familiarize yourself with the mechanics of source code compilation, and you'll end up with a smaller customized kernel with only the capabilities you need. The `gcc` compiler and accompanying tools must be installed in order to compile a kernel; these tools are a near-necessity on a Linux system even if you're not a programmer. The YAST dependency checking will help insure that all of the required compilation tools are installed.

Kernel compilation can seem daunting to a beginner, but it is a fairly intuitive process. Three interfaces are available for the initial configuration step. The first (and oldest) is a console-mode script invoked via the command `make config`. This script asks a series of questions and uses the results to write a file which guides the compiler in its work. You need to know some basic facts about your hardware such as what type of hard disk and CD-ROM drive you have. If you want sound support you'll need to know the IRQ your card uses, as well as a few other parameters that can be gathered from the card's manual or the output of the MS-DOS `msd` utility.

The other two interfaces are `menuconfig` and `xconfig`. The first uses a modified version of the `dialog` program mentioned above, which runs on a virtual console or a `xterm` and resembles the YAST setup tool. `xconfig` is a Tk-based version, designed to run in a X window. All three accomplish the same task. The latter two let you make choices without typing much. The kernel sources are well-documented. The `README` file in the top-level directory contains enough information to nearly guarantee a successful build.

2.7.3 Getting X up and running.

Successfully configuring the X Window System (specifically XFree86, which is included with S.u.S.E. and most other distributions) can be a stumbling block. There is such a multiplicity of monitors and video cards that each installation of X must be individually configured. The difficulty has been eased somewhat with the release of XFree86 3.2, which is included with the most recent S.u.S.E. release. A `dialog` based configuration tool can now be used in place of the previous `xf86config`. Both are based on shell scripts similar to the one that is used to configure the Linux kernel. Nonetheless, you will still need to know your monitor's horizontal and vertical refresh rates as well as the chip set installed on your video card. It helps to initially set your sites low, i.e., get X functioning at a low resolution first before attempting to make full use of your video card's capabilities.

The S.u.S.E. developers have taken some pains in configuring the various window managers, for example, `fvwm95`. The first time you start X, many of the applications you elected for installation will be available from the mouse activated root window menu. Another entry on the menu allows you to change the window background.

Many well-designed icons are supplied with the S.u.S.E. distribution. This gives new users something of a reprieve. After getting Linux and X running finally, there is enough to do just learning the system without feeling compelled to customize the environment, in order to make it tolerable to view!

2.7.4 Later upgrades.

The minute you finish installing even the most up-to-date distribution, it begins to incrementally become outdated. This is a slow process, but eventually you will feel the need to upgrade some part of the system. with S.u.S.E. 5.1, YaST can now update via ftp.

2.8 Post-installation procedures.

At this point it's a good idea to explain how to reboot and shutdown the system as you're using it. You should never reboot or shutdown your Linux system by pressing the reset switch. You shouldn't simply switch off the power, either. As with most UNIX systems, Linux caches disk writes in memory. Therefore, if you suddenly reboot the system without shutting down "cleanly", you can corrupt the data on your drives, causing untold damage.

The easiest way to shut down the system is with the `shutdown` command. As an example, to shutdown and reboot the system immediately, use the following command as `root`:

```
# shutdown --r now
```

This cleanly reboots your system. The manual page for `shutdown` describes the other command-line arguments that are available. Use the command `man shutdown` to see the manual page for `shutdown`.

Note, however, that many Linux distributions do not provide the `shutdown` command on the installation media. This means that the first time you reboot your system after installation, you may need to use the `Ctrl`-`Alt`-`Del` combination.

After you have a chance to explore and use the system, there are several configuration chores that you should undertake. The first is to create a user account for yourself (and, optionally, any other users that might have access to the system). Creating user accounts

is described in Chapter 4. Usually, all that you have to do is `login` as `root`, and run the `adduser` (sometimes `useradd`) program. This leads you through several prompts to create new user accounts.

If you create more than one filesystem for Linux, or if you're using a swap partition, you may need to edit the file `/etc/fstab` in order for those filesystems to be available automatically after rebooting. If you're using a separate filesystem for `/usr`, and none of the files that should be in `/usr` appear to be present, you may simply need to mount that filesystem. See page 196 for a description of the `/etc/fstab` file.

2.9 Running into trouble.

Almost everyone runs into some kind of snag or hangup when attempting to install Linux the first time. Most of the time, the problem is caused by a simple misunderstanding. Sometimes, however, it can be something more serious, like an oversight by one of the developers, or a bug.

If your installation appears to be successful, but you received unexpected error messages, these are described here as well.

2.9.1 Problems with booting the installation media

When attempting to boot the installation media for the first time, you may encounter a number of problems. These are listed below. Note that the following problems are *not* related to booting your newly installed Linux system. See page 122 for information on these kinds of pitfalls.

- **Floppy or media error when attempting to boot.**

  The most popular cause for this kind of problem is a corrupt boot floppy. Either the floppy is physically damaged, in which case you should re-create the disk with a *brand new* floppy, or the data on the floppy is bad, in which case you should verify that you downloaded and transferred the data to the floppy correctly. In many cases, simply re-creating the boot floppy will solve your problems. Retrace your steps and try again.

  If you received your boot floppy from a mail order vendor or some other distributor, instead of downloading and creating it yourself, contact the distributor and ask for a new boot floppy—but only after verifying that this is indeed the problem.

- **System hangs during boot or after booting.**

After the installation media boots, you will see a number of messages from the kernel itself, indicating which devices were detected and configured. After this, you will usually be presented with a login prompt, allowing you to proceed with installation (some distributions instead drop you right into an installation program of some kind). The system may appear to hang during several of these steps. Be patient: loading software from floppy is comparatively slow. In many cases, the system has not hung at all but is merely taking a long time. Verify that there is no drive or system activity for at least several minutes before assuming that the system is hung.

1. After booting from the `LILO` prompt, the system must load the kernel image from floppy. This may take several seconds; you will know that things are going well if the floppy drive light is still on.
2. While the kernel boots, SCSI devices must be probed for. If you do not have any SCSI devices installed, the system will hang for up to 15 seconds while the SCSI probe continues; this usually occurs after the line

   ```
   lp_init:  lp1 exists (0), using polling driver
   ```

   appears on your screen.
3. After the kernel is finished booting, control is transferred to the system boot-up files on the floppy. Finally, you will be presented with a `login` prompt, or be dropped into an installation program. If you are presented with a login prompt such as

   ```
   Linux login:
   ```

   you should then `login` (usually as `root` or `install`—this varies with each distribution). After entering the user name, the system may pause for 20 seconds or more while the installation program or shell is being loaded from floppy. Again, the floppy drive light should be on. Don't assume that the system is hung.

Any of the above items may be the source of your problem. However, it is possible that the system actually may hang while booting, which can be due to several causes. First of all, you may not have enough available RAM to boot the installation media. (See the following item for information on disabling the ramdisk to free up memory.)

The cause of many system hangs is hardware incompatibility. The last chapter presented an overview of supported hardware under Linux. Even if your hardware is supported, you may run into problems with incompatible hardware configurations

which are causing the system to hang. See page 116, below, for a discussion of hardware incompatibilities.

- **System reports out-of-memory errors while attempting to boot or install the software.**

  This item deals with the amount of RAM that you have available. On systems with 4 megabytes of RAM or less, you may run into trouble booting the installation media or installing the software itself. This is because many distributions use a RAM disk, a file system loaded directly into RAM, for operations while using the installation media. The entire image of the installation boot floppy, for example, may be loaded into a RAM disk, which may require more than a megabyte of RAM.

  You may not see an "out of memory" error when attempting to boot or install the software; instead, the system may unexpectedly hang, or fail to boot. If your system hangs, and none of the explanations in the previous section seem to be the cause, try disabling the ramdisk. See your distribution's documentation for details.

  Keep in mind that Linux itself requires at least 2 megabytes of RAM to run at all; most modern distributions of Linux require 4 megabytes or more.

- **The system reports an error like "`permission denied`" or "`file not found`" while booting.**

  This is an indication that your installation bootup media is corrupt. If you try to boot from the installation media (and you're sure that you're doing everything correctly), you should not see any errors like this. Contact the distributor of your Linux software and find out about the problem, and perhaps obtain another copy of the boot media if necessary. If you downloaded the boot disk yourself, try re-creating it and see if this solves the problem.

- **The system reports the error "`VFS: Unable to mount root`" when booting.**

  This error message means that the root file system (found on the boot media itself), could not be found. This means that either your boot media is corrupt in some way, or that you are not booting the system correctly.

  For example, many CD-ROM distributions require that you have the CD-ROM in the drive when booting. Be sure that the CD-ROM drive is on and check for any activity. It's also possible that the system is not locating your CD-ROM drive at boot time; see page 116 for more information.

2.9.2 Hardware problems.

The most common form of problem when attempting to install or use Linux is an incompatibility with hardware. Even if all of your hardware is supported by Linux, a misconfiguration or hardware conflict can sometimes cause strange results—your devices may not be detected at boot time, or the system may hang.

It is important to isolate these hardware problems if you suspect that they may be the source of your trouble.

Isolating hardware problems If you experience a problem that you believe to be hardware-related, the first thing that you should to do is attempt to isolate the problem. This means eliminating all possible variables and (usually) taking the system apart, piece-by-piece, until the offending piece of hardware is isolated.

This is not as frightening as it may sound. Basically, you should remove all nonessential hardware from your system, and then determine which device is causing the trouble—possibly by reinserting each device, one at a time. This means that you should remove all hardware other than the floppy and video controllers, and of course the keyboard. Even innocent-looking devices such as mouse controllers can wreak unknown havoc on your peace of mind unless you consider them nonessential.

For example, let's say that the system hangs during the Ethernet board detection sequence at boot time. You might hypothesize that there is a conflict or problem with the Ethernet board in your machine. The quick and easy way to find out is to pull the Ethernet board, and try booting again. If everything goes well, then you know that either (a) the Ethernet board is not supported by Linux (see page 25), or (b) there is an address or IRQ conflict with the board.

"Address or IRQ conflict?" What on earth does that mean? All devices in your machine use an **IRQ**, or *interrupt request line*, to tell the system that they need something done on their behalf. You can think of the IRQ as a cord that the device tugs when it needs the system to take care of some pending request. If more than one device is tugging on the same cord, the kernel won't be able to detemine which device it needs to service. Instant mayhem.

Therefore, be sure that all of your installed devices use unique IRQ lines. In general, the IRQ for a device can be set by jumpers on the card; see the documentation for the particular device for details. Some devices do not require the use of an IRQ at all, but it is suggested that you configure them to use one if possible. (The Seagate ST01 and ST02 SCSI controllers are good examples).

In some cases, the kernel provided on your installation media is configured to use cer-

tain IRQs for certain devices. For example, on some distributions of Linux, the kernel is preconfigured to use IRQ 5 for the TMC-950 SCSI controller, the Mitsumi CD-ROM controller, and the bus mouse driver. If you want to use two or more of these devices, you'll need to first install Linux with only one of these devices enabled, then recompile the kernel in order to change the default IRQ for one of them. (See Chapter 4 for information on recompiling the kernel.)

Another area where hardware conflicts can arise is with DMA (direct memory access) channels, I/O addresses, and shared memory addresses. All of these terms describe mechanisms through which the system interfaces with hardware devices. Some Ethernet boards, for example, use a shared memory address as well as an IRQ to interface with the system. If any of these are in conflict with other devices, then the system may behave unexpectedly. You should be able to change the DMA channel, I/O or shared memory addresses for your various devices with jumper settings. (Unfortunately, some devices don't allow you to change these settings.)

The documentation for various hardware devices should specify the IRQ, DMA channel, I/O address, or shared memory address that the devices use, and how to configure them. Again, the simple way to get around these problems is to temporarily disable the conflicting devices until you have time to determine the cause of the problem.

The table below is a list of IRQ and DMA channels used by various "standard" devices on most systems. Almost all systems have some of these devices, so you should avoid setting the IRQ or DMA of other devices in conflict with these values.

Problems recognizing hard drive or controller. When Linux boots, you should see a series of messages on your screen such as:

```
Console:  colour EGA+ 80x25, 8 virtual consoles
Serial driver version 3.96 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty03 at 0x02e8 (irq = 3) is a 16550A
lp_init:  lp1 exists (0), using polling driver
...
```

Here, the kernel is detecting the various hardware devices present on your system. At some point, you should see the line

```
Partition check:
```

followed by a list of recognized partitions, for example:

```
Partition check:
hda:  hda1 hda2
hdb:  hdb1 hdb2 hdb3
```

If, for some reason, your drives or partitions are not recognized, then you will not be able to access them in any way.

There are several things that can cause this to happen:

- **Hard drive or controller not supported.** If you have a hard drive controller (IDE, SCSI, or otherwise) that is not supported by Linux, the kernel will not recognize your partitions at boot time.

- **Drive or controller improperly configured.** Even if your controller is supported by Linux, it may not be configured correctly. (This is particularly a problem for SCSI controllers. Most non-SCSI controllers should work fine without any additional configuration).

  Refer to the documentation for your hard drive and/or controller. In particular, many hard drives need to have a jumper set to be used as a slave drive (the second device on either the primary or secondary IDE bus). The acid test of this kind of condition is to boot MS-DOS or some other operating system that is known to work with your drive and controller. If you can access the drive and controller from another operating system, then it is not a problem with your hardware configuration.

  See page **??**, above, for information on resolving possible device conflicts, and page **??**, below, for information on configuring SCSI devices.

- **Controller properly configured, but not detected.** Some BIOS-less SCSI controllers require the user to specify information about the controller at boot time. A description of how to force hardware detection for these controllers begins on page **??**.

- **Hard drive geometry not recognized.** Some systems, like the IBM PS/ValuePoint, do not store hard drive geometry information in the CMOS memory, where Linux expects to find it. Also, certain SCSI controllers need to be told where to find drive geometry in order for Linux to recognize the layout of your drive.

  Most distributions provide a bootup option to specify the drive geometry. In general, when booting the installation media, you can specify the drive geometry at the LILO boot prompt with a command such as:

```
boot:  linux hd=cylinders,heads,sectors
```

where *cylinders*, *heads*, and *sectors* correspond to the number of cylinders, heads, and sectors per track for your hard drive.

After installing Linux, you will be able to install LILO, allowing you to boot from the hard drive. At that time, you can specify the drive geometry to LILO, making it unnecessary to enter the drive geometry each time you boot. See Chapter 4 for more information about LILO.

Problems with SCSI controllers and devices. Presented here are some of the most common problems with SCSI controllers and devices like CD-ROMs, hard drives, and tape drives. If you have problems getting Linux to recognize your drive or controller, read on.

The Linux SCSI HOWTO (see Appendix A) contains much useful information on SCSI devices in addition to that listed here. SCSI can be particularly tricky to configure at times.

- **A SCSI device is detected at all possible IDs.** This is caused by strapping the device to the same address as the controller. You need to change the jumper settings so that the drive uses a different address than the controller.
- **Linux reports sense errors, even if the devices are known to be error-free.** This can be caused by bad cables or bad termination. If your SCSI bus is not terminated at both ends, you may have errors accessing SCSI devices. When in doubt, always check your cables.
- **SCSI devices report timeout errors.** This is usually caused by a conflict with IRQ, DMA, or device addresses. Also check that interrupts are enabled correctly on your controller.
- **SCSI controllers that use BIOS are not detected.** Detection of controllers that use BIOS will fail if the BIOS is disabled, or if your controller's signature is not recognized by the kernel. See the Linux SCSI HOWTO, available from the sources in Appendix A, for more information about this.
- **Controllers using memory mapped I/O do not work.** This is caused when the memory-mapped I/O ports are incorrectly cached. Either mark the board's address space as uncacheable in the XCMOS settings, or disable cache altogether.
- **When partitioning, you get a warning that "cylinders $>$ 1024", or you are unable to boot from a partition using cylinders numbered above 1023.** BIOS limits the

number of cylinders to 1024, and any partition using cylinders numbered above this won't be accessible from the BIOS. As far as Linux is concerned, this affects only booting; once the system has booted you should be able to access the partition. Your options are to either boot Linux from a boot floppy, or boot from a partition using cylinders numbered below 1024.

- **CD-ROM drive or other removeable media devices are not recognized at boot time.** Try booting with a CD-ROM (or disk) in the drive. This is necessary for some devices.

If your SCSI controller is not recognized, you may need to force hardware detection at boot time. This is particularly important for BIOS-less SCSI controllers. Most distributions allow you to specify the controller IRQ and shared memory address when booting the installation media. For example, if you are using a TMC-8xx controller, you may be able to enter

```
boot:  linux tmx8xx=interrupt,memory-address
```

at the LILO boot prompt, where *interrupt* is the IRQ of controller, and *memory-address* is the shared memory address. Whether or not this is possible depends on the distribution of Linux; consult your documentation for details.

2.9.3 Problems installing the software.

Actually installing the Linux software should be quite trouble-free, if you're lucky. The only problems that you might experience would be related to corrupt installation media or lack of space on your Linux filesystems. Here is a list of these common problems.

- **System reports "`Read error`", "`file not found`", or other errors while attempting to install the software.** This indicates a problem with the installation media. If you install from floppy, keep in mind that floppies are quite succeptible to media errors of this type. Be sure to use brand-new, newly formatted floppies. If you have an MS-DOS partition on your drive, many Linux distributions allow you to install the software from the hard drive. This may be faster and more reliable than using floppies.

  If you use a CD-ROM, be sure to check the disc for scratches, dust, or other problems that may cause media errors.

  The cause of the problem may be that the media is in the incorrect format. Many Linux distributions require that the floppies be formatted in high-density MS-DOS

format. (The boot floppy is the exception; it is not in MS-DOS format in most cases.) If all else fails, either obtain a new set of floppies, or recreate the floppies (using new diskettes) if you downloaded the software yourself.

- **System reports errors such as "`tar: read error`" or "`gzip: not in gzip format`".** This problem is usually caused by corrupt files on the installation media. In other words, your floppy may be error-free, but the data on the floppy is in some way corrupted. If you downloaded the Linux software using text mode, rather than binary mode, then your files will be corrupt, and unreadable by the installation software.

- **System reports errors like "`device full`" while installing.** This is a clear-cut sign that you have run out of space when installing the software. Not all Linux distributions can pick up the mess cleanly; you shouldn't be able to abort the installation and expect the system to work.

  The solution is usually to re-create your file systems (with `mke2fs`) which deletes the partially installed software. You can attempt to re-install the software, this time selecting a smaller amount of software to install. In other cases, you may need to start completely from scratch, and rethink your partition and filesystem sizes.

- **System reports errors such as "`read_intr: 0x10`" while accessing the hard drive.** This usually indicates bad blocks on your drive. However, if you receive these errors while using `mkswap` or `mke2fs`, the system may be having trouble accessing your drive. This can either be a hardware problem (see page 116), or it might be a case of poorly specified geometry. If you used the

  `hd=`*cylinders*`,`*heads*`,`*sectors*

  option at boot time to force detection of your drive geometry, and incorrectly specified the geometry, you could be prone to this problem. This can also happen if your drive geometry is incorrectly specified in the system CMOS.

- **System reports errors like "`file not found`" or "`permission denied`".** This problem can occur if not all of the necessary files are present on the installation media (see the next paragraph) or if there is a permissions problem with the installation software. For example, some distributions of Linux have been known to have bugs in the installation software itself. These are usually fixed very rapidly, and are quite infrequent. If you suspect that the distribution software contains bugs,

and you're sure that you have not done anything wrong, contact the maintainer of the distribution to report the bug.

If you have other strange errors when installing Linux (especially if you downloaded the software yourself), be sure that you actually obtained all of the necessary files when downloading. For example, some people use the FTP command

```
mget *.*
```

when downloading the Linux software via FTP. This will download only those files that contain a "`.`" in their filenames; if there are any files without the "`.`", you will miss them. The correct command to use in this case is

```
mget *
```

The best advice is to retrace your steps when something goes wrong. You may think that you have done everything correctly, when in fact you forgot a small but important step somewhere along the way. In many cases, re-downloading and re-installing the software can solve the problem. Don't beat your head against the wall any longer than you have to!

Also, if Linux unexpectedly hangs during installation, there may be a hardware problem of some kind. See page 116 for hints.

2.9.4 Problems after installing Linux.

You've spent an entire afternoon installing Linux. In order to make space for it, you wiped your MS-DOS and OS/2 partitions, and tearfully deleted your copies of SimCity and Wing Commander. You reboot the system, and nothing happens. Or, even worse, *something* happens, but it's not what should happen. What do you do?

On page 113, we cover some of the most common problems that can occur when booting the Linux installation media—many of those problems may apply here. In addition, you may be victim to one of the following maladies.

Problems booting Linux from floppy. If you use a floppy to boot Linux, you may need to specify the location of your Linux root partition at boot time. This is especially true if you are using the original installation floppy itself, and not a custom boot floppy that was created during installation.

While booting the floppy, hold down `Shift` or `Ctrl`. This should present you with a boot menu. Press `Tab` to see a list of available options. For example, many distributions allow you to type

```
boot:  linux hd=partition
```

at the boot menu, where *partition* is the name of the Linux root partition, like `/dev/hda2`. Consult the documentation for your distribution for details.

Problems booting Linux from the hard drive. If you opted to install LILO instead of creating a boot floppy, you should be able to boot Linux from the hard drive. However, the automated LILO installation procedure used by many distributions is not always perfect. It may make incorrect assumptions about your partition layout, and you will need to re-install LILO to get everything correct. LILO installation is covered in Chapter 4.

- **System reports "`Drive not bootable---Please insert system disk.`"** The hard drive's master boot record is corrupt in some way. In most cases, it's harmless, and everything else on your drive is still intact. There are several ways around this:

  1. While partitioning your drive using `fdisk`, you may have deleted the partition that was marked as "active". MS-DOS and other operating systems attempt to boot the "active" partition at boot time (Linux pays no attention to whether the partition is "active" or not). You may be able to boot MS-DOS from floppy and run `FDISK.EXE` to set the active flag on your MS-DOS paritition, and all will be well.

     Another command to try (with MS-DOS 5.0 and higher) is

     ```
     FDISK /MBR
     ```

     This command attempts to rebuild the hard drive master boot record for booting MS-DOS, by overwriting LILO. If you no longer have MS-DOS on your hard drive, you need to boot Linux from floppy and attempt to install LILO later.

  2. If you created a MS-DOS partition using Linux's version of `fdisk`, or vice versa, you may get this error. You should create MS-DOS partitions only using MS-DOS's version, `FDISK.EXE`. (This applies to operating systems other than MS-DOS.) The best solution is either to start from scratch and repartition the drive correctly, or to merely delete and re-create the offending partitions with the correct version of `fdisk`.

  3. The LILO installation procedure may have failed. In this case, you should either boot from your Linux boot floppy (if you have one), or from the original installation media. Either of these should provide options for specifying the

Linux root partition to use when booting. Hold down Shift or Ctrl at boot time, and press Tab from the boot menu for a list of options.

- **When booting the system from the hard drive, MS-DOS (or another operating system) starts instead of Linux.** First of all, be sure that you actually installed LILO when installing the Linux software. If not, then the system still boots MS-DOS (or whatever other operating system you may have) when you attempt to boot from the hard drive. In order to boot Linux from the hard drive, you need to install LILO (see Chapter 4).

  On the other hand, if you *did* install LILO, and another operating system boots instead of Linux, then you have LILO configured to boot that other operating system by default. While the system is booting, hold down Shift or Ctrl, and press Tab at the boot prompt. This should present you with a list of possible operating systems to boot; select the appropriate option (usually "`linux`") to boot Linux.

  If you wish to select Linux as the default operating system, you must re-install LILO. See Chapter 4.

  It may also be possible that you attempted to install LILO, but the installation procedure failed in some way. See the previous item.

Problems logging in After booting Linux, you should be presented with a `login` prompt, like

```
linux login:
```

At this point, either the distribution's documentation or the system itself will tell you what to do. For many distributions, you simply log in as `root`, with no password. Other possible user names to try are `guest` or `test`.

Most newly installed Linux systems should not require a password for the initial log in. However, if you are asked to enter a password, there may be a problem. First, try using a password equivalent to the username; that is, if you are logging in as `root`, use "`root`" as the password.

If you simply can't log in, there may be a problem. First, consult your distribution's documentation; the user name and password to use may be buried in there somewhere. The user name and password may have been given to you during the installation procedure, or they may be printed on the `login` banner.

One cause may be a problem with installing the Linux `login` program and initialization files. You may need to reinstall (at least parts of) the Linux software, or boot your

installation media and attempt to fix the problem by hand—see Chapter 4 for hints.

Problems using the system. If logging in is successful, you should be presented with a shell prompt (like "#" or "$") and can happily roam around your system. However, there are some initial problems with using the system that sometimes creep up.

The most common initial configuration problem is incorrect file or directory permissions. This can cause the error message

```
Shell-init:  permission denied
```

to be printed after logging in (in fact, any time you see the message "`permission denied`" you can be fairly certain that it is a problem with file permissions).

In many cases, it's a simple matter of using `chmod` to fix the permissions of the appropriate files or directories. For example, some distributions of Linux once used the (incorrect) file mode 0644 for the root directory (`/`). The fix was to issue the command

```
# chmod 755 /
```

as `root`. However, in order to issue this command, you needed to boot from the installation media and mount your Linux root filesystem by hand—a hairy task for most newcomers.

As you use the system, you may run into places where file and directory permissions are incorrect, or software does not work as configured. Welcome to the world of Linux! While most distributions are quite trouble-free, very few of them are perfect. We don't want to cover all of those problems here. Instead, throughout the book we help you to solve many of these configuration problems by teaching you how to find them and fix them yourself. In Chapter 1 we discussed this philosophy in some detail. In Chapter 4, we give hints for fixing many of these common configuration problems.

RPM Category	**Required?**	**Comments**
BASE	Yes	Should not be customized.
C Development	Highly Recommend	Need the minimal system to compile a kernel.
Development Libs	Highly Recommend	Need the minimal system to compile a kernel.
C++ Development	Optional	C++ Development.
Networked Workstation	Recommend; Required for other network software	Whether you are on an Ethernet or going to dialup networking, you need to install this package suite; You shouldn't customize this.
Anonymous FTP/Gopher Server	Optional	If your Linux box is going to serve files via FTP or Gopher.
Web Server	Optional	Useful for Web Developers for local development, required if you serve Web pages.
Network Management Workstation	Optional	Has additional tools useful for dialup as well as Ethernet network.
Dialup Workstation	Recommended	Required if you are going to dialup.
Game Machine	Optional	Need I say more? Fortunes are required for humor.
Multimedia Machine	Optional	If you have supported hardware.
X Window System	Optional	If you want to run X.
X Multimedia Support	Optional	If you have supported hardware.
TEX Document Formatting	Optional	Installation of the *entire* package is recommended.
`emacs`	Recommended	The One True Editing Environment.
`emacs` with X support	Recommended	Requires X
MS-DOS and Microsoft Windows Connectivity	Optional	Huh?
Extra Documentation	Required	Manual pages should *always* be installed.

Table 2.4: Important Red Hat Linux packages.

Use of Partition	Recommend	Size Comments
Swap	2 x Physical RAM	If less than 16MB of RAM installed, 16MB of swapis a must. If space is tight, and 16MB RAM installed, 1 x Physical RAM is the minimum recommended.
Root system, no X	100 - 200MB	Depends on tools, like compilers, that are needed.
Root system, with X	250-350MB	Depends on tools like compilers, that are needed.
/home	5 - Infinite MB	Depends on being single or multiple users and needs.
/var	5 - Infinite	Depends on news feeds, number of users, etc.
/usr/local	25 - 200MB	Used for programs not in RPM format or to be kept separate from the rest of Red Hat.
/usr	350+ MB	

Table 2.5: Typical Red Hat Linux disk space requirements.

File	IDE Slackware bootdisks:
`aztech.i`	CD-ROM drives: Aztech CDA268-01A, Orchid CD-3110, Okano/Wearnes CDD110, Conrad TXC, CyCDROM CR520, CR540.
`bare.i`	IDE support only.
`cdu31a.i`	Sony CDU31/33a CD-ROM.
`cdu535.i`	Sony CDU531/535 CD-ROM.
`cm206.i`	Philips/LMS cm206 CD-ROM with cm260 adapter card.
`goldstar.i`	Goldstar R420 CD-ROM (sometimes sold in a Reveal "Multimedia Kit").
`mcd.i`	NON-IDE Mitsumi CD-ROM support.
`mcdx.i`	Improved NON-IDE Mitsumi CD-ROM support.
`net.i`	Ethernet support.
`optics.i`	Optics Storage 8000 AT CD-ROM (the "DOLPHIN" drive).
`sanyo.i`	Sanyo CDR-H94A CD-ROM support.
`sbpcd.i`	Matsushita, Kotobuki, Panasonic, CreativeLabs (Sound Blaster), Longshine and Teac NON-IDE CD-ROM support.
`xt.i`	MFM hard drive support.

Table 2.6: Slackware IDE boot disk images.

File	**SCSI/IDE Slackware bootdisks:**
`7000fast.s`	Western Digital 7000FASST SCSI support.
`Advansys.s`	AdvanSys SCSI support.
`Aha152x.s`	Adaptec 152x SCSI support.
`Aha1542.s`	Adaptec 1542 SCSI support.
`Aha1740.s`	Adaptec 1740 SCSI support.
`Aha2x4x.s`	Adaptec AIC7xxx SCSI support (For these cards: AHA-274x, AHA-2842, AHA-2940, AHA-2940W, AHA-2940U, AHA-2940UW, AHA-2944D, AHA-2944WD, AHA-3940, AHA-3940W, AHA-3985, AHA-3985W).
`Am53c974.s`	AMD AM53/79C974 SCSI support.
`Aztech.s`	All supported SCSI controllers, plus CD-ROM support for Aztech CDA268-01A, Orchid CD-3110, Okano/Wearnes CDD110, Conrad TXC, CyCDROM CR520, CR540.
`Buslogic.s`	Buslogic MultiMaster SCSI support.
`Cdu31a.s`	All supported SCSI controllers, plus CD-ROM support for Sony CDU31/33a.
`Cdu535.s`	All supported SCSI controllers, plus CD-ROM support for Sony CDU531/535.
`Cm206.s`	All supported SCSI controllers, plus Philips/LMS cm206 CD-ROM with cm260 adapter card.
`Dtc3280.s`	DTC (Data Technology Corp) 3180/3280 SCSI support.
`Eata_dma.s`	DPT EATA-DMA SCSI support. (Boards like PM2011, PM2021, PM2041, PM3021, PM2012B, PM2022, PM2122, PM2322, PM2042, PM3122, PM3222, PM3332, PM2024, PM2124, PM2044, PM2144, PM3224, PM3334.)
`Eata_isa.s`	DPT EATA-ISA/EISA SCSI support. (Boards like PM2011B/9X, PM2021A/9X, PM2012A, PM2012B, PM2022A/9X, PM2122A/9X, PM2322A/9X).
`Eata_pio.s`	DPT EATA-PIO SCSI support (PM2001 and PM2012A).
`Fdomain.s`	Future Domain TMC-16x0 SCSI support.
`Goldstar.s`	All supported SCSI controllers, plus Goldstar R420 CD-ROM (sometimes sold in a Reveal “Multimedia Kit”).
`In2000.s`	Always IN2000 SCSI support.
`Iomega.s`	IOMEGA PPA3 parallel port SCSI support (also supports the parallel port version of the ZIP drive).
`Mcd.s`	All supported SCSI controllers, plusstandard non-IDE Mitsumi CD-ROM support.
`Mcdx.s`	All supported SCSI controllers, plus enhanced non-IDE Mitsumi CD-ROM support.
`N53c406a.s`	NCR 53c406a SCSI support.
`N_5380.s`	NCR 5380 and 53c400 SCSI support.
`N_53c7xx.s`	NCR 53c7xx, 53c8xx SCSI support (Most NCR PCI SCSI controllers use this driver).
`Optics.s`	All supported SCSI controllers, plus support for the Optics Storage 8000 AT CDROM (the “DOLPHIN” drive).
`Pas16.s`	Pro Audio Spectrum/Studio 16 SCSI support.
`Qlog_fas.s`	ISA/VLB/PCMCIA Qlogic FastSCSI! support (also supports the Control Concepts SCSI cards based on the Qlogic FASXXX chip).
`Qlog_isp.s`	Supports all Qlogic PCI SCSI controllers, except the PCI-basic, which the AMD SCSI driver supports.
`Sanyo.s`	All supported SCSI controllers, plus Sanyo CDR-H94A CD-ROM support.
`Sbpcd.s`	All supported SCSI controllers, plus Matsushita, Kotobuki, Panasonic, CreativeLabs (Sound Blaster), Longshine and Teac NON-IDE CDROM support.
`Scsinet.s`	All supported SCSI controllers, plus full ethernet support.

Device	I/O address	IRQ	DMA
`ttyS0` (`COM1`)	3f8	4	n/a
`ttyS1` (`COM2`)	2f8	3	n/a
`ttyS2` (`COM3`)	3e8	4	n/a
`ttyS3` (`COM4`)	2e8	3	n/a
`lp0` (`LPT1`)	378 - 37f	7	n/a
`lp1` (`LPT2`)	278 - 27f	5	n/a
`fd0`, `fd1` (floppies 1 and 2)	3f0 - 3f7	6	2
`fd2`, `fd3` (floppies 3 and 4)	370 - 377	10	3

Table 2.8: Common device settings.

Chapter 3

Linux Tutorial

3.1 Introduction.

If you're new to UNIX and Linux, you may be a bit intimidated by the size and apparent complexity of the system before you. This chapter does not go into great detail or cover advanced topics. Instead, we want you to hit the ground running.

We assume very little here about your background, except perhaps that you have 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, Linux looks a lot like MS-DOS—after all, parts of MS-DOS were modeled on the CP/M operating system, which in turn was modeled on UNIX. However, only the most superficial features of Linux resemble MS-DOS. Even if you're completely new to the PC world, this tutorial should 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. Linux has built-in security features to prevent "normal" users from damaging files that are essential to the system. Even so, the worst thing that can happen is that you may delete some or all of your files and you'll have to re-install the system. So, at this point, you have nothing to lose.

3.2 Basic Linux concepts.

Linux is a multitasking, multiuser operating system, which means that many people can run many different applications on one computer at the same time. This differs from MS-

DOS, where only one person can use the system at any one time. Under Linux, to identify yourself to the system, you must **log in**, which entails entering your **login name** (the name the system uses to identify you), and entering your **password**, which is your personal key for logging in to your account. Because only you know your password, no one else can log in to the system under your user name.

On traditional UNIX systems, the system administrator assigns you a user name and an initial password when you are given an account on the system. However, because in Linux tt you are the system administrator, you must set up your own account before you can log in. For the following discussions, we'll use the imaginary user name, "`larry`."

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

3.2.1 Creating an account.

Before you can use a newly installed Linux system, you must set up a user account for yourself. It's usually not a good idea to use the `root` account for normal use; you should reserve the `root` account for running privileged commands and for maintaining the system as discussed below.

In order to create an account for yourself, log in as `root` and use the `useradd` or `adduser` command. See Section 4.6 for information on this procedure.

3.2.2 Logging in.

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

```
mousehouse login:
```

Enter your user name and press the Enter key. Our hero, `larry`, would type:

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

Next, enter your password. The characters you enter won't be echoed to the screen, 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 user name and password, you are officially logged in to the system, and are free to roam.

3.2.3 Virtual consoles.

The system's **console** is the monitor and keyboard connected directly to the system. (Because Linux 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 VCs), that let you have more than one login session on the console at one time.

To demonstrate this, log in to your system. Next, press `Alt-F2`. You should see the `login:` prompt again. You're looking at the second virtual console. 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 lets you to access only the first half-dozen or so VCs, by pressing `Alt-F1` through `Alt-F4`, or however many VCs are configured on your system. It is possible to enable up to 12 VCs—one for each function key on your keyboard. As you can see, use of VCs can be very powerful because you can work in several different sessions at the same time.

While the use of VCs is somewhat limiting (after all, you can look at only one VC at a time), it should give you a feel for the multiuser capabilities of Linux. While you're working on the first VC, you can switch over to the second VC and work on something else.

3.2.4 Shells and commands.

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

As soon as you log in, the system starts the shell, and you can begin entering commands. Here's a quick example. Larry logs in and is waiting at the shell **prompt**.

```
mousehouse login:  larry
Password:  larry's password
```

```
Welcome to Mousehouse!

/home/larry#
```

The last line of this text 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` is 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 the burning question: What is a command? 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. Example:

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

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

When you enter a command, the shell does several things. First, it checks the command to see if it 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 disk, having the specified name. If successful, the shell runs the program, sending the arguments specified on the command line.

In our example, the shell looks for a program called `make`, and runs it with the argument `love`. `Make` is a program often used to compile large programs, and 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 returns us to the shell prompt.

What happens if we type a command to a shell and the shell can't find a program having the specified name? Well, we can try the following:

```
/home/larry# eat dirt
```

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

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

3.2.5 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 log out. There are other ways of logging out, but this is the most foolproof one.

3.2.6 Changing your password.

You should also know how to change your password. The command `passwd` prompts you for your old password, and a new password. It also asks 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 are the system administrator, see page 201.)

3.2.7 Files and directories.

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

Files are identified by their file names. For example, the file containing your history paper might be saved with the file name `history-paper`. These names usually identify the file and its contents in some form that is meaningful to you. There is no standard format for file names as there is under MS-DOS and some other operating systems; in general, a file name can contain any character (except the `/` character—see the discussion of path names, below) and is limited to 256 characters in length.

With the concept of files comes the concept of directories. A **directory** is a collection of files. It can be thought of as a "folder" that contains many different files. Directories 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.

Consequently, you can refer to a file by its **path name**, which is made up of the filename, preceded by the name of the directory containing 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 `english-lit` file, Larry can specify the file's pathname, as in:

```
papers/english-lit
```

As you can see, the directory and filename are separated by a single slash (`/`). For this reason, filenames themselves cannot contain the `/` character. 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 there is another directory within `papers`, called `notes`. The `notes` directory contains the files `math-notes` and `cheat-sheet`. The pathname of the file `cheat-sheet` would be

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

Therefore, a path name is really like a path to the file. The directory that contains a given subdirectory is known as the **parent directory**. Here, the directory `papers` is the parent of the `notes` directory.

3.2.8 The directory tree.

Most Linux systems use 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 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 (like `cheat-sheet` and `history-final`) are contained in Larry's home directory. Usually, user home directories are contained under `/home`, and are named for the user owning that directory. Larry's home directory is `/home/larry`.

The diagram on page 137 shows a sample directory tree, which should give you an idea of how the directory tree on your system is organized.

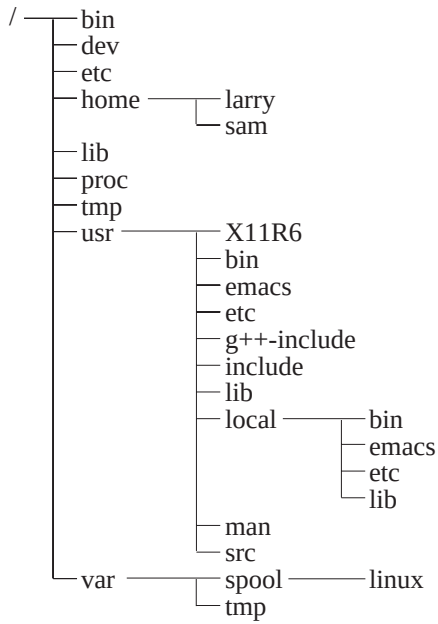


Figure 3.1: A typical (abridged) Linux directory tree.

3.2.9 The current working directory.

At any moment, commands that you enter are assumed to be relative to your **current working directory**. You can think of your working directory as the directory in which you are currently "located". When you first log in, your working directory is set to your home directory—`/home/larry`, in our case. Whenever you refer to a file, you may refer to it in relationship to your current working directory, rather than 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 by using the command

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

If you begin a filename (like `papers/final`) with a character other than `/`, you're referring to the file in terms relative to your current working directory. This is known as a **relative path name**.

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

3.2.10 Referring to home directories.

Under both `tcsh` and `bash`[1] you can specify your home directory with 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 shell replaces the `~` character with the name of your home directory.

You can also specify other user's home directories with the tilde character. The pathname `~karl/letters` translates to `/home/karl/letters` by the shell (if `/home/karl` is karl's home directory). Using a tilde is simply a shortcut; there is no directory named `~`—it's just syntactic sugar provided by the shell.

3.3 First steps into Linux.

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

[1] `tcsh` and `bash` are two *shells* that run under Linux. The shell is a program that reads user commands and executes them; most Linux systems enable either `tcsh` or `bash` for new user accounts.

3.3.1 Moving around.

Now that you can log in, and you know how to refer to files using pathnames, how can you change your current working directory, to make life easier?

The command for moving around in the directory structure is `cd`, which is short for "change directory". Many often-used Linux 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 become the current working directory.

As mentioned earlier, when you log in, you begin in your home directory. If Larry wanted to switch to 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 next higher (or parent) directory, use the command

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

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

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

gets us nowhere.

You can also use absolute pathnames with 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#
```

3.3.2 Looking at the contents of directories.

Now that you know how to move around directories, you might think, "So what?" Moving around directories is fairly useless by itself, so let's introduce a new command, `ls`. The `ls` command displays 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 of the `ls` command to get more detailed information.

```
/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 in the resulting list which would indicate 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 nor 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 the `-F` option. 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, the system 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           startcon
bcheckrc        gettydefs       motd            rc1.d           swapoff
brc             group           mount           rc2.d           swapon
brc~            inet            mtab            rc3.d           syslog.c
csh.cshrc       init            mtools          rc4.d           syslog.p
csh.login       init.d          pac             rc5.d           syslogd.
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#
```

If you're a MS-DOS user, you may notice that the filenames can be longer than 8 characters, and can contain periods in any position. You can even use more than one period in a filename.

Let's move to the top of the directory tree, and then down to another directory with the commands

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

You can also move into directories in one step, as in `cd /usr/bin`.

Try moving around various directories, using `ls` and `cd`. In some cases, you may run into the foreboding "`Permission denied`" error message. This is simply UNIX

security kicking in: in order to use the `ls` or `cd` commands, you must have permission to do so. We talk more about this starting on page 161.

3.3.3 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#
```

Congratulations! You 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.

3.3.4 Copying files.

To copy files, use the command `cp`, as shown here:

```
/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 that we use "`.`" to refer to the current directory.

3.3.5 Moving files.

The `mv` command moves files, rather than 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 that the `termcap` file has been renamed `sells`. You can also use the `mv` command to move a file to a completely new directory.

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

3.3.6 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, which stands for "remove", as shown here:

```
/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.

3.3.7 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`, 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 too quickly for you to read it all. 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 again in section 3.14.1.

3.3.8 Getting online help.

Almost every UNIX system, including Linux, provides a facility known as **manual pages**. These manual pages contain online documentation for system commands, resources, configuration files and so on.

The command used to access manual pages is `man`. If you're interested in learning about other options of the `ls` command, you can type

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

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

Unfortunately, most manual pages are written for those who already have some idea of what the command or resource does. For this reason, manual pages usually contain only the technical details of the command, without much explanation. However, manual pages can be an invaluable resource for jogging your memory if you forget the syntax of a command. Manual pages will also tell you about commands that we don't cover in this book.

I suggest that you try `man` for the commands that we've already gone over and whenever I introduce a new command. Some of these commands won't have manual pages, for several reasons. First, the manual pages may not have been written yet. (The Linux Documentation Project is responsible for manual pages under Linux as well. We are gradually accumulating most of the manual pages available for the system.) Second, the the command might be an internal shell command, or an alias (discussed on page 133), which would not have a manual page of its own. One example is `cd`, which is an internal shell command. The shell itself actually processes the `cd`—there is no separate program that implements this command.

3.4 Accessing MS-DOS files.

If, for some twisted and bizarre reason, you want to access files from MS-DOS, it's 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 file system. 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 4.8.4 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
```

mounts it. Be sure to `umount` the partition when you're done using it. You can have a MS-DOS partition automatically mounted at boot time if you include the entry in `/etc/fstab`. See Section 4.4 for details. The following line in `/etc/fstab` will mount an MS-DOS partition on `/dev/hda1` on the directory `/dos`.

```
/dev/hda1     /dos     msdos      defaults
```

You can also mount the VFAT file systems that are used by Windows 95:

```
# mount -t vfat /dev/hda1 /mnt
```

This allows access to the long filenames of Windows 95. This only applies to partitions that actually have the long filenames stored. You can't mount a normal FAT16 file system and use this to get long filenames.

The Mtools software may also be used to access MS-DOS files. The commands `mcd`, `mdir`, and `mcopy` all behave like their MS-DOS counterparts. If you install Mtools, there should be manual pages available for these commands.

Accessing MS-DOS files is one thing; running MS-DOS programs is another. There is an MS-DOS Emulator under development for Linux; it is widely available, and included in most distributions. It can also be retrieved from a number of locations, including the various Linux FTP sites listed in Appendix B. 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 limited. In addition, a Microsoft Windows emulator that runs under X Windows is under development.

3.5 Summary of basic UNIX commands.

This section introduces some of the most useful basic commands of a UNIX system, including those that are covered in the previous section.

Note that options usually begin with "`-`", and in most cases you can specify more than one option with a single "`-`". For example, rather than use the command `ls -l -F`, you can use `ls -lF`.

Rather than listing every option for each command, we only present useful or important commands at this time. In fact, most of these commands have many options that 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 as arguments a list of files or directories, denoted in this table 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*
Where *directory* is the directory which you want to change to. ("`.`" refers to the current directory, "`..`" the parent directory. If no directory is is specified it defaults you your home directory.)
Example: `cd ../foo` sets the current directory up one level, then back down to `foo`.

`ls` — Displays information about the named files and directories.
Syntax: `ls` *files*
Where *files* consists of the the filenames or directories to list. The most commonly used options are `-F` (to display the file type), and `-l` (to give a "long" listing including file size, owner, permissions, and so on).
Example: `ls -lF /home/larry` displays the contents of the directory `/home/larry`.

`cp` — Copies one or more file to another file or directory.
Syntax: `cp` *files destination*
Where *files* lists 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 one or more file to another file or directory. This command does the equivalent of a copy followed by the deletion of the original file. You can use this to rename files, like in the MS-DOS command `RENAME`.
Syntax: `mv` *files destination*
Where *files* lists 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 you delete a file under UNIX, they are unrecoverable (unlike MS-DOS, where you can usually "undelete" the file).
Syntax: `rm` *files*
Where *files* describes the filenames to delete.
The `-i` option prompts 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` *dirs*
Where *dirs* are the directories to create.
Example: `mkdir /home/larry/test` creates the directory `test` in `/home/larry`.

`rmdir` Deletes empty directories. When using `rmdir`, the current working directory must not be within the directory to be deleted.
Syntax: `rmdir` *dirs*
Where *dirs* defines the directories to delete.
Example: `rmdir /home/larry/papers` deletes the directory `/home/larry/papers`, if empty.

`man` Displays the manual page for the given command or resource (that is, any system utility that 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` *files*
Where *files* lists 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 contents of a file on screen.
Syntax: `cat` *files*
Where *files* lists the files to display.
Example: `cat letters/from-mdw` displays the file `letters/from-mdw`.

`echo` Displays the given arguments on the screen.
Syntax: `echo` *args*
Where *args* lists arguments to echo.
Example: `echo "Hello world"` displays the string "`Hello world`".

`grep` Display every line in one or more files that match the given pattern.
Syntax: `grep` *pattern files*
Where *pattern* is a regular expression pattern, and *files* lists the files to search.
Example: `grep loomer /etc/hosts` displays every line in the file `/etc/hosts` that contains the pattern "`loomer`".

3.6 Exploring the file system.

A **file system** is the collection of files and the hierarchy of directories on a system. The time has now come to escort you around the file system.

You now have the skills and the knowledge to understand the Linux file system, and you have a roadmap. (Refer to diagram on page 137).

First, change to the root directory (`cd /`), and then enter `ls -F` to display a listing of its contents. You'll probably see the following directories[2]: `bin`, `dev`, `etc`, `home`, `install`, `lib`, `mnt`, `proc`, `root`, `tmp`, `user`, `usr`, and `var`.

[2] You may see others, and you might not see all of them. Every release of Linux differs in some respects.

Now, let's take a look at each of these directories.

`/bin` `/bin` is short for "binaries", or executables, where many essential system programs reside. Use `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, for example, 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 on page 140.

`/dev` The "files" in `/dev` are **device files**—they access system devices and resources like disk drives, modems, and memory. Just as your system can read data from a file, it can also read input from the mouse by accessing `/dev/mouse`.

Filenames that begin with `fd` are floppy disk devices. `fd0` is the first floppy disk drive, and `fd1` is the second. You may have noticed that there are more floppy disk devices than the two listed above: these represent specific types of floppy disks. For example, `fd1H1440` accesses high-density, 3.5" diskettes in drive 1.

The following is a list of some of the most commonly used device files. Even though you may not have some of the physical devices listed below, chances are that you'll have drivers 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. `/dev/ttyS0` refers to "`COM1`" under MS-DOS. The `/dev/cua` devices are "callout" devices, and used with a modem.
- Device names beginning with `hd` access hard drives. `/dev/hda` refers to the *whole* first hard disk, while `/dev/hda1` refers to the first *partition* on `/dev/hda`.
- Device names that begin 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.

- Device names that begin with `lp` access parallel ports. `/dev/lp0` is the same as "`LPT1`" in the MS-DOS world.
- `/dev/null` is used as a "black hole"—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.
- Devices whose names are `/dev/tty` followed by a number refer to the "virtual consoles" on your system (accessed 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.
- Device names beginning with `/dev/pty` are **pseudo-terminals**, which 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 a number of miscellaneous system configuration files. These include `/etc/passwd` (the user database), `/etc/rc` (the system initialization script), and so on.

`/sbin` `/sbin` contains essential system binaries that are used for system administration.

`/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**, which are files that contain code which many programs share in common. Rather than each program using 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` supports a "virtual file system", where the files are stored in memory, not on disk. These "files" refer to the various **processes** running on the system, and let you get information about the programs and

processes that are running at any given time. This is discussed in more detail starting on page 166.

`/tmp` Many programs store temporary information and in a file that is deleted when the program has finished executing. The standard location for these files is in `/tmp`.

`/usr` `/usr` is a very important directory which contains subdirectories that 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 items found in `/usr` are optional. However, it is these optional items that make the system useful and interesting. Without `/usr`, you'd have a boring system that supports only programs like `cp` and `ls`. `/usr` contains most of the larger software packages and the configuration files that accompany them.

`/usr/X11R6` `/usr/X11R6` contains The X Window System, if you installed it. The X Window System is a large, powerful graphical environment that 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 familiar. The `/usr/X11R6` directory contains all of the X Windows executables, configuration files, and support files. This is covered in more detail in Chapter **??**.

`/usr/bin` `/usr/bin` is the real warehouse for software on any Linux system, containing most of the executables for programs not found in other places, like `/bin`.

`/usr/etc` Just as `/etc` contains essential miscellaneous system programs and configuration files, `/usr/etc` contains miscellaneous utilities and files, that in general, are not essential to the system.

`/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. Files 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 like `stdio.h`, which declare functions like `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 for 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 much like `/usr`—it contains various programs and files not essential to the system, but which make the system fun and exciting. In general, programs in `/usr/local` are specialized for your system—consequently, `/usr/local` differs greatly between Linux systems.

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

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

`/var`
: `/var` holds directories that often change in size or tend to grow. Many of those directories used to reside in `/usr`, but since those who support Linux are trying to keep it relatively unchangeable, the directories that change often have been moved to `/var`. Some Linux distributions maintain their software package databases in directories under `/var`.

`/var/log`
: `/var/log` contains various files of interest to the system administra-

tor, specifically system logs, which record 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 4.

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

3.7 Types of shells.

As mentioned before, Linux is a multitasking, multiuser operating system. Multitasking is *very* useful, and once you understand it, you'll use it all of the time. Before long, you'll run programs in the background, switch between tasks, and pipeline programs together to achieve complicated results with a single command.

Many of the features we'll cover in this section are features provided by the shell itself. Be careful not to confuse Linux (the actual operating system) with a shell—a shell is just an interface to the underlying system. The shell provides functionality inaddition to Linux itself.

A shell is not only an interpreter for the interactive commands you type at the prompt, but also a powerful programming language. It lets you to write **shell scripts**, to "batch" several shell commands together in a file. If you know MS-DOS you'll recognize the similarity to "batch files". Shell scripts are a very powerful tool, that will let you automate and expand your use of Linux. See page 181 for more information.

There are several types of shells in the Linux 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, like System III. The name of the Bourne shell on most Linux 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 Linux systems is named `/bin/csh`.

Under Linux, several variations of these shells are 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 that includes many of the advanced features found in the C shell. Because `bash` supports a superset of the Bourne shell syntax, shell scripts written in the standard Bourne shell should work with `bash`. If you prefer to use the C shell syntax, Linux supports `tcsh`, which is an expanded version of the original C shell.

The type of shell 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 shell you use doesn't matter—the syntax is the same. Only when you start to write shell scripts or use advanced features of a shell do the differences between shell types begin to matter.

As we discuss the features of the various shells, we'll note 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`).

3.8 Wildcards.

A key feature of most Linux shells is the ability to refer to more than one file by name using special characters. These **wildcards** let you refer to, say, all file names that contain the character "`n`".

The wildcard "`*`" specifies any character or string of characters in a file name. When you use the character "`*`" in a file name, the shell replaces it with all possible substitutions from file names in the directory you're referencing.

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

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

To specify all files containing the letter "o" in the filename, use the command

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

As you can see, each instance of "`*`" is replaced with all substitutions that match 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 a series of filenames is called **wildcard expansion** and is done by the shell. This is important: an individual command, such as `ls`, *never* sees the "`*`" in its list of parameters. The shell expands the wildcard to include all filenames that match. So, the command

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

is expanded by the shell to

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

One important note about the "`*`" wildcard: it does *not* match file names that 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 mentioned earlier that each directory contains two special entries: "`.`" 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 that begin with "`.`". Observe:

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

The listing contains 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. They are described starting on page 185.

Note that when you 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 expands to only a single character. Thus, "`ls ?`" displays 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 lets you specify many files at one time. In the command summary that starts on page 146, we said that the `cp` and `mv` commands actually can copy or move more than one file at a time. For example,

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

copies all filenames in `/etc` beginning with "`s`" to the directory `/home/larry`. The format of the `cp` command is really

`cp` *files destination*

where *files* lists the filenames to copy, and *destination* is the destination file or directory. `mv` has an identical syntax.

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.

3.9 Linux plumbing.

3.9.1 Standard input and standard output.

Many Linux commands get input from what is called **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 `cat` command. Normally, `cat` reads data from all of the files specified by the command line, and sends this data directly to `stdout`. Therefore, using the command

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

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

However, if you don't specify a filename, `cat` 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#
```

Each line that you type is immediately echoed back by `cat`. When reading from standard input, you indicate the input is "finished" by sending an EOT (end-of-text) signal, in general, generated by pressing `Ctrl-D`.

Here's another example. The `sort` command reads lines of text (again, from stdin, unless you specify one or more filenames) 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 Linux useful?

3.9.2 Redirecting input and output.

Now, let's say that you want to send the output of `sort` to a file, to save our shopping list on disk. The shell lets you **redirect** standard output to a filename, by 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, but is saved to the file named `shopping-list`. Let's look at this file:

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

Now you can sort your shopping list, and save it, too! But let's suppose that you are storing the 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 follows:

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

However, there's another way to do this. Not only can you redirect standard output, you 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 lets you demonstrate the following 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 still reads 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 that reads data from standard input, processes it in some way, and sends the processed data to standard output. Using redirection, standard input and standard output can be referenced from files. As mentioned above, `stdin` and `stdout` default to the keyboard and screen respectively. `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 is given to it.

3.9.3 Using pipes.

We already demonstrated how to use `sort` as a filter. However, these examples assume that you have data stored in a file somewhere or are willing to type the data from the standard input yourself. What if the data that you wanted to sort came from the output of another command, like `ls`?

The `-r` option to `sort` sorts the data in reverse-alphabetical order. If you want to list the files in your current directory in reverse order, one way to do it is follows:

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

Now redirect the output of the `ls` command into a file called `file-list`:

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

```
notes
masters-thesis
history-final
english-list
/home/larry/papers#
```

Here, you save the output of `ls` in a file, and then run `sort -r` on that file. But this is unwieldy and uses a temporary file to save the data from `ls`.

The solution is **pipelining**. This is a shell feature that connects a string of commands via a "pipe." The `stdout` of the first command is sent to the `stdin` of the second command. In this case, we want to send the `stdout` of `ls` to the `stdin` of `sort`. Use the "`|`" symbol to create a pipe, as follows:

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

This command is shorter and easier to type.

Here's another useful example, the command

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

displays a long list of files, most of which fly past the screen too quickly for you to read. So, 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 leisure.

But the fun doesn't stop here! You can pipe more than two commands together. The command `head` is a filter that displays the first lines from an input stream (in this case, input from a pipe). If you want to display the last filename in alphabetical order in the current directory, use commands like the following:

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

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

3.9.4 Non-destructive redirection of output.

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 is appended to (added to the end of) the named file instead of overwriting it. For example,

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

appends the output of the `ls` command to `file-list`.

Keep in mind that redirection and pipes are features of the shell—which supports the use of "`>`", "`>>`" and "`|`". It has nothing to do with the commands themselves.

3.10 File permissions.

3.10.1 Concepts of file permissions.

Because there is typically more than one user on a Linux system, Linux provides a mechanism known as **file permissions**, which protect user files from tampering by other users. This mechanism lets files and directories be "owned" by a particular user. For example, because Larry created the files in his home directory, Larry owns those files and has access to them.

Linux also lets files be shared between users and groups of users. If Larry desired, he could cut off access to his files so 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.

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

Groups are usually defined by the type of users who access the machine. For example, on a university Linux system users may be placed into the groups `student`, `staff`, `faculty` or `guest`. There are also a few system-defined groups (like `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 lets a user read the contents of the file, or in the case of directories, list the contents of the directory (using `ls`). Write permission lets the user write to and modify the file. For directories, write permission lets the user create new files or delete files within that directory. Finally, execute permission lets the user run the file as a program or shell script (if the file is a program or shell script). For directories, having execute permission lets the user `cd` into the directory in question.

3.10.2 Interpreting file permissions.

Let's look at an example that demonstrates file permissions. Using the `ls` command with the `-l` option displays a "long" listing of the file, including file permissions.

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

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

/home/larry/foo#
```

The first field 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`). We'll cover the other fields later.

This file is owned by `larry`, and belongs to the group `users`. 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 "`-`" means that this is a regular file (as opposed to a directory or device driver). The next three characters ("`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 mentioned, besides read and write permission, there is also "execute" permission—represented by an "`x`". However, a "`-`" is listed here in place of an "`x`", so Larry doesn't have execute permission on this file. This is fine, as 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 that owns this file is `users`. Because only an "`r`" appears here, any user who 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 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.

3.10.3 Permissions Dependencies.

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 set the permissions to 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 (or execute) access to the file itself.

Typically, users on a Linux system are very open with their files. The usual set of permissions given to files is `-rw-r--r--`, which lets other users read the file but not change it in any way. The usual set of permissions given to directories is `-rwxr-xr-x`, which lets other users 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 prevent any other user from accessing the file. Likewise, setting the permissions of a directory to `-rwx------` keeps other users out of the directory in question.

3.10.4 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`
: Let the owner of the file 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.

3.11 Managing file links.

Links let you give a single file more than one name. Files are actually identified by the system by their **inode number**, which is just the unique file system identifier for the file. 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.

3.11.1 Hard links.

The `ln` command is used to create multiple links for one file. For example, let's say that you have a file called `foo` in a directory. Using `ls -i`, you can look at the inode number for this file.

```
/home/larry# ls -i foo
22192 foo
/home/larry#
```

Here, `foo` has an inode number of 22192 in the file system. You can create another link to `foo`, named `bar`, as follows:

```
/home/larry# ln foo bar
```

With `ls -i`, you see that the two files have the same inode.

```
/home/larry# ls -i foo bar
22192 bar   22192 foo
/home/larry#
```

Now, specifying either `foo` or `bar` will access the same file. If you make changes to `foo`, those changes appear in `bar` as well. For all purposes, `foo` and `bar` are the same file.

These links are known as **hard links** because they create a direct link to an inode. Note that you can hard-link files only when they're on the same file system; 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

```
/home/larry# rm foo
```

then only the link named `foo` is deleted, `bar` will still exist. A file is only truly 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 delete only a single link; in order to delete the file, you must delete all links to the file.

The command `ls -l` displays the number of links to a file (among other information).

```
/home/larry# ls -l foo bar
-rw-r--r--   2 root     root          12 Aug  5 16:51 bar
-rw-r--r--   2 root     root          12 Aug  5 16:50 foo
/home/larry#
```

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 associations. Also, every directory contains at least two hard links: "`.`" (a link pointing to itself), and "`..`" (a link pointing to the parent directory). The root directory (`/`) "`..`" link just points back to `/`. (In other words, the parent of the root directory is the root directory itself.)

3.11.2 Symbolic links.

Symbolic links, or **symlinks**, are another type of link, which are different from hard links. A symbolic link lets you give a file another name, but doesn't link the file by inode.

The command `ln -s` creates a symbolic link to a file. For example, if you use the command

```
/home/larry# ln -s foo bar
```

you will create a symbolic link named `bar` that points to the file `foo`. If you use `ls -i`, you'll see that the two files have different inodes, indeed.

```
/home/larry# ls -i foo bar
22195 bar   22192 foo
/home/larry#
```

However, using `ls -l`, we see that the file `bar` is a symlink pointing to `foo`.

```
/home/larry# ls -l foo bar
lrwxrwxrwx   1 root     root            3 Aug  5 16:51 bar -> foo
-rw-r--r--   1 root     root           12 Aug  5 16:50 foo
/home/larry#
```

The file permissions 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 differences. For one thing, you can create a symbolic link to a file that doesn't exist; the same is not true for hard links. Symbolic links are processed by the kernel differently than are hard links, which is just a technical difference but sometimes an important one. Symbolic links are helpful because they identify the 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 page 223 for more information.

3.12 Job control.

3.12.1 Jobs and processes.

Job control is a feature provided by many shells (including `bash` and `tcsh`) that

let you 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 called a process. The command `ps` displays a list of currently running processes, as shown here:

```
/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 looking only at the processes which Larry himself is currently running. (There are many other processes running on the system as well—"`ps -aux`" lists them all.) 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` has 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 also called a *job*. 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 that lets you switch between several independent jobs.

In most cases users run only a single job at a time—whatever command they last typed to the shell. However, using job control, you can run several jobs at once, and switch between them as needed.

How might this be useful? Let's say you are editing a text file and want to interrupt your editing and do something else. With job control, you can temporarily suspend the editor, go back to the shell prompt and start to work on something else. When you're done, you can switch back to the editor and be back where you started, as if you didn't leave the editor. There are many other practical uses of job control.

3.12.2 Foreground and background.

Jobs can either be in the **foreground** or in the **background**. There can only be one job in the foreground at a time. The foreground job is the job with which you interact—it receives input from the keyboard and sends output to your screen, unless, of course, you

have redirected input or output, as described starting on page 157). On the other hand, jobs in the background do not receive input from the terminal—in general, they run along quietly without the 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; just run them in the background. While jobs run in the background, you are free to run other programs.

Jobs may also be **suspended**. A suspended job is a job that 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 does not change the state of the job in any way—the job continues to run where it left off.

Suspending a job is not equal to interrupting a job. When you **interrupt** a running process (by pressing the interrupt key, which is usually `Ctrl-C`)[3], the process is killed, for good. Once the job is killed, there's no hope of resuming it. You'll must run the command again. Also, some programs trap the interrupt, so that pressing `Ctrl-C` won't immediately kill the job. This is to let the program perform any necessary cleanup operations before exiting. In fact, some programs don't let you kill them with an interrupt at all.

3.12.3 Backgrounding and killing jobs.

Let's begin with a simple example. The command `yes` is a seemingly useless command that 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 pressing the interrupt key, which is usually `Ctrl-C`. So that we don't have to put up with the annoying stream of

[3] You can set the interrupt key with the `stty` command.

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 disappears. 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, press the interrupt key.

Let's suppose that you want the `yes` command to continue to run but wanted to get the shell prompt back so that you can work on other things. You can put `yes` into the background, allowing it to run, 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#
```

As you can see, the shell prompt has returned. 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 we're running, it is assigned job number `1`. The "`164`" is the process ID, or PID, number given by the system to the job. You can use either number to refer to the job, as you'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 internal shell 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 `kill` command. 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
```

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

Now that you've killed the job, 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 you use the `jobs` command again nothing should be printed.

You can also kill the job using the process ID (PID) number, displayed 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.

3.12.4 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.

First, start the `yes` process in the foreground, as you did before:

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

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

Now, rather than interrupt the job with `Ctrl-C`, **suspend** the job. Suspending a job doesn't kill it: it only temporarily stops the job until you restart it. To do this, press 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 causes 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 `fg` command (for "foreground").

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

The shell displays 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 `bg` command to put the job into the background. This causes 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 you have your prompt back. `Jobs` should report that `yes` is indeed running, and you can kill the job with `kill` as we did before.

How can you 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 that 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 have been 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 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 displayed on your screen, without any way for you to interrupt it. (You can't use Ctrl-C to interrupt jobs in the background.) In order to stop the endless `y`'s, use the `fg` command to bring the job to the foreground, and then use Ctrl-C to kill it.

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

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

(to put job number 2 into the foreground), or

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

(to put job number 3 into the background). 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 `fg`, `bg` and `jobs` commands are internal to the shell. If for some reason you use a shell that doesn't support job control, don't expect to find these commands available.

In addition, there are some aspects of job control that differ between `bash` and `tcsh`. In fact, some shells don't provide job control at all—however, most shells available for Linux do.

3.13 Using the `vi` editor.

A **text editor** is a program used to edit files that are composed of text: a letter, C program, or a system configuration file. While there are many such editors available for Linux, the only editor that you are guaranteed to find on any UNIX or Linux system is `vi`— the "visual editor." `vi` is not the easiest editor to use, nor is it very self-explanatory. However, because `vi` is so common in the UNIX/Linux world, and sometimes necessary, it deserves discussion here.

Your choice of an editor 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 other 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 artificial intelligence program). However, because Emacs and its support files are relatively large, it may not be installed on some 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.

This section presents an introduction to `vi`—we won't discuss all of its features, only the ones you need to know to get started. You can refer to the man page for `vi` if you're interested in learning more about this editor's features. Alternatively, you can read the book

Learning the vi Editor from O'Reilly and Associates, or the *VI Tutorial* from Specialized Systems Consultants (SSC) Inc. See Appendix A for information.

3.13.1 Concepts.

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

When you start up `vi`, you are in *command mode*. This mode lets you use commands to edit files or 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 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 in this mode. You start insert mode by using a command such as "`i`" (for "insert") from command mode. While in insert mode, you can insert text into the document at the 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 "`:`" in command mode, you jump into last line mode and can use commands like "`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 that are longer than one character. In last line mode, you enter a single-line command and press `Enter` to execute it.

3.13.2 Starting `vi`.

The best way to understand these concepts is to fire up `vi` and edit a file. The example "screens" below show only a few lines of text, as if the screen were only six lines high instead of twenty-four.

The syntax for `vi` is

```
vi filename
```

where *filename* is the name of the file to edit.

Start up `vi` by typing

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

to edit the file `test`. You should see something like

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

The column of "~" characters indicates you are at the end of the file. The _ represents the cursor.

3.13.3 Inserting text.

The `vi` program is now in command mode. Insert text into the file by pressing `i`, which places the editor into insert mode, and begin typing.

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

Type as many lines as you want (pressing `Enter` after each). You may correct mistakes with the `Backspace` key.

To end insert mode and return to command mode, press `Esc`.

In command mode you can use the arrow keys to move around in the file. (If you have only 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 the `i` command. The `a` command inserts text beginning after the current cursor position, instead of at 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 press [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 next line, use the o command. Press [o] and enter another line or two:

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

3.13.4 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 humans to come to the
aid of the party.
Afterwards, we'll go out for pizza and_
~
~
~
~
```

Now press a and insert some text, followed by esc:

```
Now is the time for all good humans 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 the cursor is on the second line and you type dd, you'll see:

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

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

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

3.13.5 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 humans to come to the aid of
the hungry.
~
~
~
~
~
```

Using `R` to edit text is like the `i` and `a` commands, but `R` overwrites, rather than inserts, text.

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

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

The "`~`" command changes the case of the letter under the cursor from upper- to lowercase, and back. 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.
~
~
~
~
~
```

3.13.6 Commands for moving the cursor.

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` command moves it to the beginning of the previous word.

The `0` command (that's the zero key) 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.

To move the cursor to the end of the file, press `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 those for deleting text. For example, the `d$` command deletes everything from the cursor to the end of the line; `dG` deletes everything from the cursor to the end of the file, and so on.

3.13.7 Saving files and quitting `vi`.

To quit `vi` without making changes to the file, use the command `:q!`. When you press the "`:`", the cursor moves to the last line on the screen and 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`. If the file has not been changed since the last save, it merely exits, preserving the modification time of the last change. Remember that you must press `Enter` after a command entered in last line mode.

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

3.13.8 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 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.

3.13.9 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
```

inserts the contents of the file `foo.txt` in the text at the location of the cursor.

3.13.10 Running shell commands.

You can also run shell commands within `vi`. The `:r!` command works like `:r`, but rather than read 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 you're editing. If you use the command

```
:shell
```

`vi` starts an instance of the shell, letting you temporarily put vi "on hold" while you execute other commands. Just log out of the shell (using the `exit` command) to return to `vi`.

3.13.11 Getting `vi` help.

`vi` doesn't provide much in the way of interactive help (most Linux programs don't), but you can always read the man page for `vi`. `vi` is a visual front-end to the `ex` editor; 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.

3.14 Customizing your environment.

A shell provides many mechanisms to customize your work environment. As mentioned above, a shell is more than a command interpreter—it is also a powerful programming language. Although 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 Linux system by using these advanced features of the shell.

As 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 encountering many differences between the two, but we will assume that shell scripts are executed using the Bourne shell syntax.

3.14.1 Shell scripts.

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

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

concatenates the files `chapter1`, `chapter2`, and `chapter3` and places the result in the file `book`. The second command displays a count of the number of lines in `book`, and the third command `lp book` prints `book`.

Rather than type all these commands, you can group them into a **shell script**. The shell script used to run all these commands might look like this:

```
#!/bin/sh
# A shell script to create and print the book
cat chapter1 chapter2 chapter3 > book
wc -l book
lp book
```

Shell scripts are just plain text files; you can create them with an editor such as `emacs` or `vi`, which is described starting on page 172.

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 Linux systems, `/bin/sh` is a Bourne-type shell, like `bash`. By forcing the shell script to run using `/bin/sh`, you ensure that the script will run under a Bourne-syntax shell (rather than 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 and make the script easier to understand.

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, make sure that you have execute permission on the script in order to run it. When you create text files, the default permissions usually don't include execute permission, and you must set them explicitly. See the discussion of file permissions on page 161 for details. Briefly, if this script were saved in the file called `makebook`, you could use the command

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

to give yourself execute permission for the shell script `makebook`.

You can use the command

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

to run all the commands in the script.

3.14.2 Shell variables and the environment.

A shell lets you define **variables**, as do most programming languages. A variable is just a piece of data that is given a name.

◇ `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 like `bash`. See the `tcsh` manual 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 then refer to this value by the variable name prefixed with a "`$`" character. For example, 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, which means that only the shell can access them. 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 `set` command displays a list of all defined shell variables.

However, the shell lets you **export** variables to the **environment**. The environment is the set of variables that are accessible by all commands that you execute. Once you define a variable inside the shell, exporting it makes the variable part of the environment as well. Use the `export` command to export a variable to the environment.

◇ Again, here we differ between `bash` and `tcsh`. If you use `tcsh`, another syntax is used for setting environment variables (the `setenv` command is used). See the `tcsh` manual page for more information.

The environment is very important to the UNIX system. It lets you 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 and it specifies the command to use to display manual pages one screenful at a time. If you set PAGER to the name of a command, it uses that command to display the man pages, instead of `more` (which is the default).

Set PAGER to "`cat`". This causes output from `man` to be displayed to the screen all at once, without pausing between 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 is 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.

It is often necessary to quote strings in order to prevent the shell from treating various characters as special. For example, you need to quote a string in order to prevent the shell from interpreting the special meaning of characters such as "*", "?" or a space. There are many other characters that may need to be protected from interpretation. A detailed explanation and desription of quoting is described in SSC's *Bourne Shell Tutorial*.

The manual pages for a particular command 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 (which contains the current working directory followed by a "#" symbol),

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

The `bash` manual page describes the syntax used for setting the prompt.

The `PATH` environment variable. When you use the `ls` command, how does the shell find the `ls` executable itself? In fact, `ls` is in `/bin` on most systems. The shell uses the environment variable `PATH` to locate executable files for commands 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 time, and means that you don't have to remember where all the command executables are stored. On many systems, executables are scattered about in many places, such as `/usr/bin`, `/bin`, or `/usr/local/bin`. Rather than give the command's full pathname (such as `/usr/bin/cp`), you can 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 lets you 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 in your `PATH`, then the shell will not search it for commands to run; this also includes the current directory.

3.14.3 Shell initialization scripts.

In addition to the 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 **initialization scripts**, which are scripts executed by the shell when you log in.

The initialization scripts themselves are simply shell scripts. However, they initialize your environment by executing commands automatically when you log in. If you always use the `mail` command to check your mail when you log in, you place the command in the initialization script so it will execute automatically.

Both `bash` and `tcsh` distinguish between a **login shell** and other invocations of the shell. A `login` shell is a shell invoked when you log in. Usually, it's the only shell you'll use. However, if you "shell out" of another program like `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 and 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/csh.login` (executed by all `tcsh` users at `login` time), `$HOME/.tcshrc` (executed at login time and by all new instances of `tcsh`), and `$HOME/.login` (executed at `login` time, following `.tcshrc`). If `.tcshrc` is not present, `.cshrc` is used instead.

A complete guide to shell programming would be beyond the scope of this book. See the manual pages for `bash` or `tcsh` to learn more about customizing the Linux environment.

3.15 So you want to strike out on your own?

This chapter should give you enough information for basic Linux use. The manual pages are indispensable tools for learning about Linux. They may appear confusing at first, but if you dig beneath the surface, there is a wealth of information.

We also suggest that you read a general Linux reference book. Linux has more features than first meet the eye. Unfortunately, many of them are beyond the scope of this book. Other recommended Linux books are listed in Appendix A.

Chapter 4

System Administration

This chapter covers the most important things that you need to know about system administration under Linux in sufficient detail to start using the system comfortably. In order to keep the chapter manageable, it covers just the basics and omits many important details. The *Linux System Administrator's Guide*, by Lars Wirzenius (see Appendix A) provides considerably more detail on system administration topics. It will help you understand better how things work and hang together. At least, skim through the SAG so that you know what it contains and what kind of help you can expect from it.

4.1 The `root` account.

Linux differentiates between different users. What they can do to each other and the system is regulated. File permissions are arranged so that normal users can't delete or modify files in directories like `/bin` and `/usr/bin`. Most users protect their own files with the appropriate permissions so that other users can't access or modify them. (One wouldn't want anybody to be able to read one's love letters.) Each user is given an **account** that includes a user name and home directory. In addition, there are special, system defined accounts which have special privileges. The most important of these is the **root account**, which is used by the system administrator. By convention, the system administrator is the user, `root`.

There are no restrictions on `root`. He or she can read, modify, or delete any file on the system, change permissions and ownerships on any file, and run special programs like those which partition a hard drive or create file systems. The basic idea is that a person who

cares for the system logs in as `root` 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.

If a normal user tries inadvertently to delete all of the files in `/etc`, the system will not permit him or her to do so. However, if `root` tries to do the same thing, the system doesn't complain at all. It is very easy to trash a Linux system when using `root`. The best way to prevent accidents is:

- Sit on your hands before you press `Enter` for any command that is non-reversible. If you're about to clean out a directory, re-read the entire command to make sure that it is correct.
- Use a different prompt for the `root` account. `root`'s `.bashrc` or `.login` file should set the shell prompt to something different than the standard user prompt. Many people reserve the character "#" in prompts for `root` and use the prompt character "$" for everyone else.
- Log in as `root` only when absolutely necessary. When you have finished your work as `root`, log out. The less you use the `root` account, the less likely you are to damage the system. You are less likely to confuse the privileges of `root` with those of a normal user.

Picture the `root` account as a special, magic hat that gives you lots of power, with which you can, by waving your hands, destroy entire cities. It is a good idea to be a bit careful about what you do with your hands. Because it is easy to wave your hands in a destructive manner, it is not a good idea to wear the magic hat when it is not needed, despite the wonderful feeling.

We'll talk in greater detail about the system administrator's responsibilities starting on page 201.

4.2 Booting the system.

Some people boot Linux with a floppy diskette that contains a copy of the Linux kernel. This kernel has the Linux root partition coded into it, so it knows where to look for the root file system. This is the type of floppy created by Slackware during installation, for example.

To create your own boot floppy, locate the kernel image on your hard disk. It should be in the file `/vmlinuz`, or `/vmlinux`. In some installations, `/vmlinuz` is a soft link to the actual kernel, so you may need to track down the kernel by following the links.

Once you know where the kernel is, set the root device of 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 `/vmlinuz` to `/dev/hda2`, use the command

```
# rdev /vmlinuz /dev/hda2
```

`rdev` can set other options in the kernel, like the default SVGA mode to use at boot time. The command

```
# rdev -h
```

prints a help message on the screen. After setting the root device, simply copy the kernel image to the floppy. Before copying data to any floppy, however, it's a good idea to use the MS-DOS `FORMAT.COM` or the Linux `fdformat` program to format the diskette. This lays down the sector and track information that is appropriate to the floppy's capacity.

Floppy diskette formats and their device driver files are discussed further starting on page 211.

Device driver files, as mentioned earlier, reside in the `/dev` directory. To copy the kernel in the file `/etc/Image` to the floppy in `/dev/fd0`, use the command

```
# cp /vmlinuz /dev/fd0
```

This floppy should now boot Linux.

4.2.1 Using LILO.

LILO is a separate boot loader which resides on your hard disk. It is executed when the system boots from the hard drive and can automatically boot Linux from a kernel image stored there.

LILO can also be used as a first-stage boot loader for several operating systems, which allows you to select the operating system you to boot, like Linux or MS-DOS. With LILO, the default operating system is booted unless you press `Shift` during the boot-up sequence, or if the `prompt` directive is given in the `lilo.conf` file. In either case, you will be provided with a boot prompt, where you type the name of the operating system to boot (such as "`linux`" or "`msdos`"). If you press `Tab` at the boot prompt, a list of operating systems that the system knows about will be provided.

The easy way to install LILO is to edit the configuration file, `/etc/lilo.conf`. The command

```
# /sbin/lilo
```

rewrites the modified `lilo.conf` configuration to the boot sector of the hard disk, and must be run every time you modify `lilo.conf`.

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. The `lilo.conf` file below is for a system which has a Linux root partition on `/dev/hda1` and a 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

# Set a sane videomode
vga = normal

# Set the delay in milli-seconds. This is the time you have to
# press the 'SHIFT' key to bring up the LILO: prompt if you
# haven't specified the 'prompt' directive.
delay = 60

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

# This forces LILO to prompt you for the OS you want to boot.
# A 'TAB' key at the LILO: prompt will display a list of the OSs
# available to boot according to the names given in the 'label='
# directives below.
prompt

# Have LILO perform some optimization.
compact

# Stanza for Linux root partition on /dev/hda1.
image = /vmlinuz     # Location of kernel
```

```
    label = linux     # Name of OS (for the LILO boot menu)
    root = /dev/hda1  # Location of root partition
    read-only         # Mount read only

# 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 is the default operating system for LILO to boot. 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 at boot time.

The Microsoft Windows '95 installer will overwrite the LILO boot manager. If you are going to install Windows '95 on your system after installing LILO, make sure to create a boot disk first (see Section 4.2). With the boot disk, you can boot Linux and re-install LILO after the Windows '95 installation is completed. This is done simply by typing, as root, the command `/sbin/lilo`, as in the step above. Partitions with Windows '95 can be configured to boot with LILO using the same `lilo.conf` entries that are used to boot the MS-DOS partition.

The *Linux FAQ* (see Appendix A) provides more information on LILO, including how to use LILO to boot with the OS/2 Boot Manager.

4.3 Shutting down.

Shutting down a Linux system can be tricky. You should never simply turn off the power or press the reset switch. The kernel keeps track of the disk read/write data in memory buffers. If you reboot the system without giving the kernel a chance to write its buffers to disk, you can corrupt the file systems.

Other precautions are taken during shutdown as well. All processes are sent a signal that allows them to die gracefully (by first writing and closing all files, for example). File systems are unmounted for safety. If you wish, the system can also alert users that the system is going down and give them a chance to log off.

The easiest way to shut down is with the `shutdown` command. The format of the command is

`shutdown` *time warning-message*

The *time* argument is the time to shut down 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 shut down immediately. The `-r` option may be given to `shutdown` to reboot the system after shutting down.

For example, to shut down and reboot the system at 8:00 pm, 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 and turn off the machine.

◇ 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 shut down the system, "cleanly," using the `shutdown` or `halt` command. On some systems, pressing `Ctrl`-`Alt`-`Del` will be trapped and cause a `shutdown`. On other systems, using the "Vulcan nerve pinch" will reboot the system immediately and cause disaster.

4.3.1 The `/etc/inittab` file.

Immediately after Linux boots and the kernel mounts the root file system, the first program that the system executes is `init`. This program is responsible for starting the system startup scripts, and modifies the system operatiing from its initial boot-up state to its standard, multiuser state. `init` also spawns the `login:` shells for all of the tty devices on the system, and specifies other startup and shutdown procedures.

After startup, `init` remains quietly in the background, monitoring and if necessary altering the running state of the system. There are many details that the `init` program must see to. These tasks are defined in the `/etc/inittab` file. A sample `/etc/inittab` file is shown below.

◇ Modifying the `/etc/inittab` file incorrectly can prevent you from logging in to your system. At the very least, when changing the `/etc/inittab` file, keep on hand a copy of the original, correct file, and a boot/root emergency floppy in case you make a mistake.

```
#
# inittab
This file describes how the INIT process should set up
# the system in a certain run-level.
```

```
#
# Version: @(#)inittab 2.04 17/05/93 MvS
#                                        2.10    02/10/95        PV
#
# Author:
Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
# Modified by: Patrick J. Volkerding, <volkerdi@ftp.cdrom.com>
# Minor modifications by:
# Robert Kiesling, <kiesling@terracom.net>
#
# Default runlevel.
id:3:initdefault:

# System initialization (runs when system boots).
si:S:sysinit:/etc/rc.d/rc.S

# Script to run when going single user (runlevel 1).
su:1S:wait:/etc/rc.d/rc.K

# Script to run when going multi user.
rc:23456:wait:/etc/rc.d/rc.M

# What to do at Ctrl-Alt-Del
ca::ctrlaltdel:/sbin/shutdown -t5 -rfn now

# Runlevel 0 halts the system.
l0:0:wait:/etc/rc.d/rc.0

# Runlevel 6 reboots the system.
l6:6:wait:/etc/rc.d/rc.6

# What to do when power fails (shutdown to single user).
pf::powerfail:/sbin/shutdown -f +5 "THE POWER IS FAILING"

# If power is back before shutdown, cancel the running shutdown.
pg:0123456:powerokwait:/sbin/shutdown -c "THE POWER IS BACK"

# If power comes back in single user mode, return to multi user mode.
ps:S:powerokwait:/sbin/init 5
```

```
# The getties in multi user mode on consoles an serial lines.
#
# NOTE NOTE NOTE adjust this to your getty or you will not be
#                able to login !!
#
# Note: for 'agetty' you use linespeed, line.
# for 'getty_ps' you use line, linespeed and also use 'gettydefs'
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
c2:1235:respawn:/sbin/agetty 38400 tty2 linux
c3:1235:respawn:/sbin/agetty 38400 tty3 linux
c4:1235:respawn:/sbin/agetty 38400 tty4 linux
c5:1235:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux

# Serial lines
# s1:12345:respawn:/sbin/agetty -L 9600 ttyS0 vt100
s2:12345:respawn:/sbin/agetty -L 9600 ttyS1 vt100

# Dialup lines
d1:12345:respawn:/sbin/agetty -mt60 38400,19200,9600,2400,1200 ttyS0 vt100
#d2:12345:respawn:/sbin/agetty -mt60 38400,19200,9600,2400,1200 ttyS1 vt100

# Runlevel 4 used to be for an X-window only system, until we discovered
# that it throws init into a loop that keeps your load avg at least 1 all
# the time. Thus, there is now one getty opened on tty6. Hopefully no one
# will notice. ;^)
# It might not be bad to have one text console anyway, in case something
# happens to X.
x1:4:wait:/etc/rc.d/rc.4

# End of /etc/inittab
```

At startup, this `/etc/inittab` starts six virtual consoles, a `login:` prompt on the modem attached to `/dev/ttyS0`, and a `login:` prompt on a character terminal connected via a RS-232 serial line to `/dev/ttyS1`.

Briefly, `init` steps through a series of **run levels**, which correspond to various operationing states of the system. Run level 1 is entered immediately after the system boots,

run levels 2 and 3 are the normal, multiuser operation modes of the system, run level 4 starts the X Window System via the X display manager `xdm`, and run level 6 reboots the system. The run level(s) associated with each command are the second item in each line of the `/etc/inittab` file.

For example, the line

```
s2:12345:respawn:/sbin/agetty -L 9600 ttyS1 vt100
```

will maintain a `login` prompt on a serial terminal for runlevels 1–5. The "`s2`" before the first colon is a symbolic identifier used internally by `init`. `respawn` is an `init` keyword that is often used in conjunction with serial terminals. If, after a certain period of time, the `agetty` program, which spawns the terminal's `login:` prompt, does not receive input at the terminal, the program times out and terminates execution. "`respawn`" tells `init` to re-execute `agetty`, ensuring that there is always a `login:` prompt at the terminal, regardless of whether someone has logged in. The remaining parameters are passed directly to `agetty` and instruct it to spawn the `login` shell, the data rate of the serial line, the serial device, and the terminal type, as defined in `/etc/termcap` or `/etc/terminfo`.

The `/sbin/agetty` program handles many details related to terminal I/O on the system. There are several different versions that are commonly in use on Linux systems. They include `mgetty`, `psgetty`, or simply, `getty`.

In the case of the `/etc/inittab` line

```
d1:12345:respawn:/sbin/agetty -mt60
38400,19200,9600,2400,1200 ttyS0 vt100
```

which allows users to log in via a modem connected to serial line `/dev/ttyS0`, the `/sbin/agetty` parameters "`-mt60`" allow the system to step through all of the modem speeds that a caller dialing into the system might use, and to shut down `/sbin/agetty` if there is no connection after 60 seconds. This is called **negotiating** a connection. The supported modem speeds are enumerated on the command line also, as well as the serial line to use, and the terminal type. Of course, both of the modems must support the data rate which is finally negotiated by both machines.

Many important details have been glossed over in this section. The tasks that `/etc/inittab` maintains would comprise a book of their own. For further information, the manual pages of the `init` and `agetty` programs, and the Linux Documentation Project's Serial HOWTO, available from the sources listed in Appendix A, are starting points.

4.4 Managing file systems.

Another task of the system administrator is caring for file systems. Most of this job entails periodically checking the file systems for damage or corrupted files. Many Linux systems also automatically check the file systems at boot time.

4.4.1 Mounting file systems.

Before a file system is accessible to the system, it must be **mounted** on a directory. For example, if you have a file system on a floppy, you must mount it under a directory like `/mnt` in order to access the files on the floppy (see page 215). After mounting the file system, all of the files in the file system appear in that directory. After unmounting the file system, the directory (in this case, `/mnt`) will be empty.

The same is true of file systems on the hard drive. The system automatically mounts file systems on your hard drive at bootup time. The so-called "root file system" is mounted on the directory `/`. If you have a separate file system for `/usr`, it is mounted on `/usr`. If you only have a root file system, all files (including those in `/usr`) exist on that file system.

`mount` and `umount` (not *unmount*) are used to mount and unmount file systems. The command

```
mount -av
```

is executed automatically by the file `/etc/rc` at boot time, or by the file `/etc/rc.d/boot` (see page 225) on some Linux systems. The file `/etc/fstab` provides information on file systems and mount points. An example `/etc/fstab` file is

```
# 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, `device`, is the name of the partition to mount. The second field is the mount point. The third field is the file system type, like `ext2` (for ext2fs) or `minix` (for Minix file systems). Table 4.1 lists the various file system types that are mountable under Linux.[1] Not all of these file system types may be available on your system, because the kernel must have support for them compiled in. See page 216 for information on building the kernel.

[1] This table is current as of kernel version 2.0.33.

File system	Type name	Comment
Second Extended File system	`ext2`	Most common Linux file system.
Extended File system	`ext`	Superseded by `ext2`.
Minix File system	`minix`	Original Minix file system; rarely used.
Xia File system	`xia`	Like `ext2`, but rarely used.
UMSDOS File system	`umsdos`	Used to install Linux on an MS-DOS partition.
MS-DOS File system	`msdos`	Used to access MS-DOS files.
`/proc` File system	`proc`	Provides process information for `ps`, etc.
ISO 9660 File system	`iso9660`	Format used by most CD-ROMs.
Xenix File system	`xenix`	Used to access files from Xenix.
System V File system	`sysv`	Used to access files from System V variants for the x86.
Coherent File system	`coherent`	Used to access files from Coherent.
HPFS File system	`hpfs`	Read-only access for HPFS partitions (DoubleSpace).

Table 4.1: Linux File system Types

The last field of the `fstab` file are the `mount` options. This is normally set to `defaults`.

Swap partitions are included in the `/etc/fstab` file. They have a mount directory of `none`, and type `swap`. The `swapon -a` command, which is executed from `/etc/rc` or `/etc/init.d/boot`, is used to enable swapping on all of the swap devices that are listed in `/etc/fstab`.

The `/etc/fstab` file contains one special entry for the `/proc` file system. As described on page 166, the `/proc` file system is used to store information about system processes, available memory, and so on. If `/proc` is not mounted, commands like `ps` will not work.

◇ The `mount` command may be used only by root. This ensures security on the system. You wouldn't want regular users mounting and unmounting file systems on a whim. Several software packages are available which allow non-root users to mount and unmount file systems, especially floppies, without compromising system security.

The `mount -av` command actually mounts all of the file systems other than the root file system (in the table above, `/dev/hda2`). The root file system is automatically mounted at boot time by the kernel.

Instead of using `mount -av`, you can mount a file system by hand. The command

```
# mount -t ext2 /dev/hda3 /usr
```

is equivalent to mounting the file system with the entry for `/dev/hda3` in the example `/etc/fstab` file, above.

4.4.2 Device driver names.

In addition to the partition names listed in the `/etc/fstab` file, Linux recognizes a number of fixed and removable media devices. They are classified by type, interface, and the order they are installed. For example, the first hard drive on your system, if it is an IDE or older MFM hard drive, is controlled by the device driver pointed to by `/dev/hda`. The first partition on the hard drive is `/dev/hda1`, the second partition is `/dev/hda2`, the third partition is `/dev/hda3`, and so on. The first partition of the second IDE drive is often `/dev/hdb1`, the second partition `/dev/hdb2`, and so on. The naming scheme for the most commonly installed IDE drives for Intel-architecture, ISA and PCI bus machines, is given in Table 4.2.

Device driver	Drive
`/dev/hda`	Master IDE drive, primary IDE bus.
`/dev/hdb`	Slave IDE drive, primary IDE bus.
`/dev/hdc`	Master IDE drive, secondary IDE bus.
`/dev/hdd`	Slave IDE drive, secondary IDE bus.

Table 4.2: IDE device driver names.

CD-ROM and tape drives which use the extended IDE/ATAPI drive interface also use these device names.

Many machines, however, including high-end personal computer workstations, and machines based on Digital Equipment Corporation's Alpha processor, use the Small Computer System Interface (SCSI). The naming conventions for SCSI devices are somewhat different than that given above, due the greater flexibility of SCSI addressing. The first SCSI hard drive on a system is `/dev/sda`, the second SCSI drive is `/dev/sdb`, and so on. A list of common SCSI devices is given in Table 4.3.

Note that SCSI CD-ROM and tape drives are named differently than SCSI hard drives. Removable SCSI media, like the Iomega Zip drive, follow naming conventions for non-removable SCSI drives. The use of a Zip drive for making backups is described starting on page 213

Streaming tape drives, like those which read and write QIC-02, QIC-40, and QIC-80 format magnetic tapes, have their own set of device names, which are described on

Device driver	Drive
`/dev/sda`	First SCSI hard drive.
`/dev/sdb`	Second SCSI hard drive.
`/dev/st0`	First SCSI tape drive.
`/dev/st1`	Second SCSI tape drive.
`/dev/scd0`	First SCSI CD-ROM drive.
`/dev/scd1`	Second SCSI CD-ROM drive.

Table 4.3: SCSI device drivers

page 214.

Floppy disk drives use still another naming scheme, which is described on page 211.

4.4.3 Checking file systems.

It is usually a good idea to check your file systems for damaged or corrupted files every now and then. Some systems automatically check their file systems at boot time (with the appropriate commands in `/etc/rc` or `/etc/init.d/boot`).

The command used to check a file system depends on the type of the file system. For ext2fs file systems (the most commonly used type), this command is `e2fsck`. For example, the command

```
# e2fsck -av /dev/hda2
```

checks the ext2fs file system on `/dev/hda2` and automatically corrects any errors.

It is usually a good idea to unmount a file system before checking it, and necessary, if `e2fsck` is to perform any repairs on the file system. The command

```
# umount /dev/hda2
```

unmounts the file system on `/dev/hda2`. The one exception is that you cannot unmount the root file system. In order to check the root file system when it's unmounted, you should use a maintenance boot/root diskette (see page 227). You also cannot unmount a file system if any of the files which it contains are "busy"—that is, in use by a running process. For example, you cannot unmount a file system if any user's current working directory is on that file system. You will instead receive a "`Device busy`" error message.

Other file system types use different forms of the `e2fsck` command, like `efsck` and `xfsck`. On some systems, you can simply use the command `fsck`, which automatically determines the file system type and executes the appropriate command.

◇ If e2fsck reports that it performed repairs on a mounted file system, you *must* reboot the system immediately. You should give the command `shutdown -r` to perform the reboot. This allows the system to re-synchronize the information about the file system after `e2fsck` modifies it.

The `/proc` file system never needs to be checked in this manner. `/proc` is a memory file system and is managed directly by the kernel.

4.5 Using a swap file.

Instead of reserving a separate partition for swap space, you can use a swap file. However, you need to install Linux and get everything running before you create the swap file.

With Linux installed, you can use the following commands to create a swap file. The command below creates a swap file of size 8208 blocks (about 8 Mb).

```
# dd if=/dev/zero of=/swap bs=1024 count=8208
```

This command creates the swap file, `/swap`. The "`count=`" parameter is the size of the swap file in blocks.

```
# mkswap /swap 8208
```

This command initializes the swap file. Again, replace the name and size of the swapfile with the appropriate values.

```
# sync
# swapon /swap
```

Now the system is swapping on the file `/swap`. The `sync` command ensures that the file has been written to disk.

One major drawback to using a swap file is that all access to the swap file is done through the file system. This means the blocks which make up the swap file may not be contiguous. Performance may not be as good as a swap partition, where the blocks are always contiguous and I/O requests are made directly to the device.

Another drawback of large swap files is the greater chance that the file system will be corrupted if something goes wrong. Keeping the regular file systems and swap partitions separate prevents this from happening.

Swap files can be useful if you need to use more swap space temporarily. If you're compiling a large program and would like to speed things up somewhat, you can create a temporary swap file and use it in addition to the regular swap space.

To remove a swap file, first use `swapoff`, as in

```
# swapoff /swap
```

Then the file can be deleted.

```
# rm /swap
```

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.

4.6 Managing users.

Even if you're the only user on your system, it's important to understand the aspects of user management under Linux. You should at least have an account for yourself (other than `root`) to do most of your work.

Each user should have his or her own account. It is seldom a good idea to have several people share the same account. Security an issue, and accounts uniquely identify users to the system. You must be able to keep track of who is doing what.

4.6.1 User management concepts.

The system keeps track of the following information about each user:

user name This identifier is unique for every user. Example user names are `larry`, `karl`, and `mdw`. Letters and digits may be used, as well as "`_`" and "`.`" (period). User names are usually limited to 8 characters in length.

user ID This number, abbreviated UID, is unique for every user. The system generally keeps track of users by UID, not user name.

group ID This number, abbreviated GID, is the user's default group. In Section 3.10, we discuss group permissions. Each user belongs to one or more groups as defined by the system administrator.

password This is 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 user name. For example, the user `schmoj` may be "Joe Schmo" in real life.

home directory
: This is the directory the user is initially placed in at login, and where his or her personal files are stored. Every user is given a home directory, which is commonly located under `/home`.

login shell
: The shell that is started for the user at login. Examples are `/bin/bash` and `/bin/tcsh`.

This information is stored in the file `/etc/passwd`. Each line in the file has the format

```
user name:encrypted password:UID:GID:full name:home
directory:login shell
```

An example might be

```
kiwi:Xv8Q981g71oKK:102:100:Laura
Poole:/home/kiwi:/bin/bash
```

In this example, the first field, "`kiwi`," is the user name.

The next field, "`Xv8Q981g71oKK`", is the encrypted password. Passwords are not stored on the system in human-readable format. The password is encrypted using itself as the secret key. In other words, one must know the password in order to decrypt it. This form of encryption is reasonably secure.

Some systems use "shadow passwords," in which password information is stored in the file `/etc/shadow`. Because `/etc/passwd` is world-readable, `/etc/shadow` provides some degree of extra security because its access permissions are much more restricted. Shadow passwords also provide other features, like password expiration.

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 is stored in the file `/etc/group`. See Section 4.6.5 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 user name. It simply helps identify the directory.

4.6.2 Adding users.

When adding users, several steps must be taken. First, the user is given an entry in `/etc/passwd`, with a unique user name and UID. The GID, full name, and other infor-

mation 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 installed in the home directory, and other files must be configured system-wide (for example, a spool for the user's incoming e-mail).

It is not difficult to add users by hand, but 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 updates all of the system files automatically. The name of this program is `useradd` or `adduser`, depending on what software is installed.

The `adduser` command takes its information from the file `/etc/adduser.conf`, which defines a standard, default configuration for all new users.

A typical `/etc/adduser.conf` file is shown below.

```
# /etc/adduser.conf: 'adduser' configuration.
# See adduser(8) and adduser.conf(5) for full documentation.

# The DSHELL variable specifies the default login shell on your
# system.
DSHELL=/bin/bash

# The DHOME variable specifies the directory containing users' home
# directories.
DHOME=/home

# If GROUPHOMES is "yes", then the home directories will be created as
# /home/groupname/user.
GROUPHOMES=no

# If LETTERHOMES is "yes", then the created home directories will have
# an extra directory - the first letter of the user name. For example:
# /home/u/user.
LETTERHOMES=no

# The SKEL variable specifies the directory containing "skeletal" user
# files; in other words, files such as a sample .profile that will be
# copied to the new user's home directory when it is created.
SKEL=/etc/skel

# FIRST_SYSTEM_UID to LAST_SYSTEM_UID inclusive is the range for UIDs
# for dynamically allocated administrative and system accounts.
```

```
FIRST_SYSTEM_UID=100
LAST_SYSTEM_UID=999

# FIRST_UID to LAST_UID inclusive is the range of UIDs of dynamically
# allocated user accounts.
FIRST_UID=1000
LAST_UID=29999

# The USERGROUPS variable can be either "yes" or "no". If "yes" each
# created user will be given their own group to use as a default, and
# their home directories will be g+s. If "no", each created user will
# be placed in the group whose gid is USERS_GID (see below).
USERGROUPS=yes

# If USERGROUPS is "no", then USERS_GID should be the GID of the group
# 'users' (or the equivalent group) on your system.
USERS_GID=100

# If QUOTAUSER is set, a default quota will be set from that user with
# 'edquota -p QUOTAUSER newuser'
QUOTAUSER=""
```

In addition to defining preset variables that the `adduser` command uses, `/etc/adduser.conf` also specifies where default system configuration files for each user are located. In this example, they are located in the directory `/etc/skel`, as defined by the `SKEL=` line, above. Files which are placed in this directory, like a system-wide, default `.profile`, `.tcshrc`, or `.bashrc` file, will be automatically installed in a new user's home directory by the `adduser` command.

4.6.3 Deleting users.

Deleting users can be accomplished with the commands `userdel` or `deluser`, depending on the software installed on the system.

If you'd like to temporarily "disable" a user from logging in to the system without deleting his or her account, 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
```

prevents `kiwi` from logging in.

4.6.4 Setting user attributes.

After you have created a user, you may need to change attributes for that user, like the 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 `passwd`. The command

```
# passwd larry
```

will change `larry`'s password. Only `root` may change other users' passwords in this manner. Users can change their own passwords, however.

On some systems, the commands `chfn` and `chsh` allow users to set their own full name and login shell attributes. If not, the system administrator must change these attributes for them.

4.6.5 Groups.

As mentioned above, 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 3.10, 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, like `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 like `users`. You can also maintain several groups for users, like `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 user names 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.

The "password" field of `/etc/group` is sometimes used to set a password on group access. This is seldom necessary. To protect users from changing into privileged 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`.

4.6.6 System administration responsibilities.

Because the system administrator has so much power and responsibility, when some users have their first opportunity to login as `root`. either on a Linux system or elsewhere, the tendency is to abuse `root`'s privileges. I have known so-called "system administrators" who read other users' mail, delete users' files without warning, and generally behave like children when given such a powerful "toy".

Because the administrator has such power on the system, it takes a certain amount of maturity and self-control to use the `root` 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 his or her users. I can't stress that enough.

4.6.7 Coping with users.

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. Ninety percent of the time, when a user is

causing trouble on the system (for instance, by filling up the user partition with large files, or running multiple instances of a large program), the user is simply unaware that he or she is creation 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 cause potential trouble, don't be accusatory. The burden of proof is on you; that is, the rule of "innocent until proven guilty" still holds. It is best to simply talk to the user and question him or her about the trouble instead of being confrontational. The last thing you want is to 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 you want to be in.

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

4.6.8 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 Linux is not designed for such discipline. It makes sense to lay down a few simple and flexible guidelines. The fewer rules you have, the less chance there is of breaking them. Even if your rules are perfectly reasonable and clear, users will still at times break them without intending to. This is especially true of new users learning the ropes of the system. It is not patently obvious that you shouldn't download a gigabyte of files and mail them to everyone on the system. Users need help to understand the rules and why they are there.

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

4.6.9 What it all means.

We don't tell you how to run your system down to the last detail. That depends on how you're using the system. If you have many users, things are much different than if you have

only 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 a Linux wizard. There are many administrators who know very little about Linux. Likewise, many "normal" users know more about Linux than any system administrator. Also, being the system administrator does not allow one to use malice against users. Just because the system gives administrators the ability to mess with user files does not mean that he or she has a right to do so.

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

4.7 Archiving and compressing files.

Before we can talk about backups, we need to introduce the tools used to archive files on UNIX systems.

4.7.1 Using `tar`.

The `tar` command is most often used to archive files. Its command syntax is

`tar` *options files*

where *options* is the list of commands and options for `tar`, and *files* is the list of files to add or extract from the archive.

For example, the command

```
# tar cvf backup.tar /etc
```

packs 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. `v` forces `tar` to use verbose mode, printing each file name 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.

◇ Old files with the same name are overwritten when extracting files into an existing directory.

Before extracting tar files it is important to know where the files should be unpacked. Let's say that you archive 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 file name. In order to extract the files to the correct location, use

```
# cd /
# tar xvf backup.tar
```

because files are extracted with the path name saved in the archive file.

However, if you archive the files with the command

```
# cd /etc
# tar cvf hosts group passwd
```

the directory name is not saved in the archive file. Therefore, you 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
```

can be used to display a listing of the archive's files without extracting them. You can see what directory the files in the archive are stored relative to, and extract the archive in the correct location.

4.7.2 `gzip` and `compress`.

Unlike archiving programs for MS-DOS, `tar` does not automatically compress files as it archives them. If you are archiving two, 1-megabyte files, the resulting tar file is two megabytes in size. The `gzip` command compresses a file (it need not be a tar file). The command

```
# gzip -9 backup.tar
```

compresses `backup.tar` and leaves you with `backup.tar.gz`, a 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, including a software patent dispute against the `compress` data compression algorithm, and the fact that `gzip` is much more efficient, `compress` is being phased out.

Files that are output by `compress` end in "`.Z`." `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. It is equivalent to "`compress -d`." `gunzip` knows how to handle `compress`ed files as well.

4.7.3 Putting them together.

To archive a group of files and compress the result, use the commands:

```
# tar cvf backup.tar /etc
# gzip -9 backup.tar
```

The result is `backup.tar.gz`. To unpack this file, use the reverse commands:

```
# gunzip backup.tar.gz
# tar xvf backup.tar
```

Always make sure that you are in the correct directory before unpacking a tar file.

You can use some Linux cleverness to do this on one command line.

```
# tar cvf - /etc | gzip -9c > backup.tar.gz
```

Here, we send the tar file to "`-`", which stands for `tar`'s standard output. This is piped to `gzip`, which compresses the incoming tar file. The result is saved in `backup.tar.gz`. The `-c` option tells `gzip` to send its output to standard output, which is redirected to `backup.tar.gz`.

A single command 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 standard output. This is piped to `tar`, which reads "`-`", this time referring to `tar`'s standard input.

[2] For some time, the extension `.z` (lowercase "`z`") was used for gzipped files. The conventional `gzip` extension is now `.gz`.

Happily, the `tar` command also includes the `z` option to automatically compress/uncompress files on the fly, using the `gzip` compression algorithm.

The command

```
# tar cvfz backup.tar.gz /etc
```

is equivalent to

```
# tar cvf backup.tar /etc
# gzip 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 `tar` and `gzip` manual pages for more information.

4.8 Using floppies and making backups.

Floppies are often 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).

As mentioned earlier, floppy diskettes must be formatted with the MS-DOS `FORMAT.COM` or the Linux `fdformat` program. This lays down the sector and track information that is appropriate to the floppy's capacity.

A few of the device names and formats of floppy disks which are accessible by Linux are given in Table 4.4.

Devices which begin with `fd0` are the first floppy diskette drive, which is named the `A:` drive under MS-DOS. The driver file names of second floppy device begin with `fd1`. Generally, the Linux kernel can detect the format of a diskette that has already been formatted—you can simply use `/dev/fd0` and let the system detect the format. But when you first use completely new, unformatted floppy disks, you may need to use the driver specification if the system can't detect the diskette's type.

A complete list of Linux devices and their device driver names is given in *Linux Allocated Devices*, by H. Peter Anvin (see Appendix A).

Floppy device driver	Format
/dev/fd0d360	Double density, 360 Kb, 5.25 inch.
/dev/fd0h1200	High density, 1.2 MB, 5.25 inch.
/dev/fd0h1440	High density, 1.44 MB, 3.5 inch.

Table 4.4: Linux floppy disk formats.

You can also use floppies to hold individual file systems and `mount` the floppy to access the data on it. See section 4.8.4.

4.8.1 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 span multiple volumes; that is, when one floppy is full, `tar` will prompt for the next. The command

```
# tar xvfzM /dev/fd0
```

restores the complete backup. This method can also be used with a tape drive connected to your system. See section 4.8.3.

Several other programs exist for making multiple-volume backups; the `backflops` program found on `tsx-11.mit.edu` may come in handy.

Making a complete backup of the system with floppies can be time- and resource-consuming. Many system administrators use an **incremental backup** policy. Every month, a complete backup is made, 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 is useful for locating files which were modified after a certain date. Several scripts for managing incremental backups can be found on `sunsite.unc.edu`.

4.8.2 Backups with a Zip drive.

Making backups to a Zip drive is similar to making floppy backups, but because Zip disks commonly have a capacity of 98 Kb, it is feasible to use a single, mounted Zip disk for a single backup archive.

Zip drives are available with three different hardware interfaces: a SCSI interface, an IDE interface and a parallel port PPA interface. Zip drive support is not included as a pre-compiled Linux option, but it can be specified when building a custom kernel for your system. Page 219 describes the installation of an Iomega Zip device driver.

The SCSI and PPA interface Zip drives use the SCSI interface and follow the naming conventions for other SCSI devices, which are described on page 198.

Zip disks are commonly pre-formatted with a MS-DOS file system. You can either use the existing MS-DOS filesystem, which must be supported by your Linux kernel, or use `mke2fs` or a similar program to write a Linux file system to the disk.

A Zip disk, when mounted as the first SCSI device, is `/dev/sda4`.

```
# mount /dev/sda4 /mnt
```

It is often convenient to provide a separate mount point for Zip file systems; for example, `/zip`. The following steps, which must be executed as `root`, would create the mount point:

```
# mkdir /zip
# chmod 0755 /zip
```

Then you can use `/zip` for mounting the Zip file system.

Writing archives to Zip disks is similar to archiving to floppies. To archive and compress the `/etc` directory to a mounted Zip drive, the command used would be

```
# tar zcvf /zip/etc.tgz /etc
```

This command could be executed from any directory because it specifies absolute path names. The archive name `etc.tgz` is necessary if the Zip drive contains a MS-DOS file system, because any files written to the disk must have names which conform to MS-DOS's 8+3 naming conventions; otherwise, the file names will be truncated.

Similarly, extracting this archive requires the commands

```
# cd /
# tar zxvf /zip/etc.tgz
```

To create, for example, an ext2 file system on a Zip drive, you would give the command (for an *unmounted* Zip disk)

```
# mke2fs /dev/sda4
```

With a Zip drive mounted in this manner, with an ext2 file system, it is possible to back up entire file systems with a single command.

```
# tar zcvf /zip/local.tar.gz /usr/local
```

Note that backing up with `tar` is still preferable in many cases to simply making an archival copy with the `cp -a` command, because `tar` preserves the original files' modification times.

4.8.3 Making backups to tape devices.

Archiving to a streaming tape drive is similar to making a backup to a floppy file system, only to a different device driver. Tapes are also formatted and handled differently that floppy diskettes. Some representative tape device drivers for Linux systems are listed in Table 4.5.

Tape device driver	Format
`/dev/rft0`	QIC-117 tape, rewind on close.
`/dev/nrft0`	QIC-117 tape, no rewind on close.
`/dev/tpqic11`	QIC-11 tape, rewind on close.
`/dev/ntpqic11`	QIC-11 tape, no rewind on close.
`/dev/qft0`	Floppy tape drive, rewind on close.
`/dev/nqft0`	Floppy tape drive, no rewind on close.

Table 4.5: Tape device drivers.

Floppy tape drives use the floppy drive controller interface and are controlled by the ftape device driver, which is covered below. Installation of the ftape device driver module is described on page 221. SCSI tape devices are listed in Table 4.3.

To archive the `/etc` directory a tape device with `tar`, use the command

```
# tar cvf /dev/qft0 /etc
```

Similarly, to extract the files from the tape, use the commands

```
# cd /
# tar xvf /dev/qft0
```

These tapes, like diskettes, must be formatted before they can be used. The ftape driver can format tapes under Linux. To format a QIC-40 format tape, use the command

```
# ftformat --format-parameter qic40-205ft --mode-auto
--omit-erase --discard-header
```

Other tape drives have their own formatting software. Check the hardware documentation for the tape drive or the documentation of the Linux device driver associated with it.

Before tapes can be removed from the drive, they must be rewound and the I/O buffers written to the tape. This is analogous to unmounting a floppy before ejecting it, because the tape driver also caches data in memory. The standard UNIX command to control tape drive operations is `mt`. Your system may not provide this command, depending on whether it has tape drive facilities. The ftape driver has a similar command, `ftmt`, which is used to control tape operations.

To rewind a tape before removing it, use the command

```
# ftmt -f /dev/qft0 rewoffl
```

Of course, substitute the correct tape device driver for your system.

It is also a good idea to retension a tape after writing to it, because magnetic tapes are susceptible to stretch. The command

```
# ftmt -f /dev/qft0 retension
```

To obtain the status of the tape device, with a formatted tape in the drive, give the command

```
# ftmt -f /dev/qft0 status
```

4.8.4 Using floppies as file systems.

You can create a file system on a floppy as you would on a hard drive partition. For example,

```
# mke2fs /dev/fd0 1440
```

creates a file system on the floppy in `/dev/fd0`. The size of the file system 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. It is necessary to specify the size of the file system in blocks if the system cannot automatically detect the floppy’s capacity.

In order to access the floppy, you must mount the file system contained on it. The command

```
# mount /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 **mount point**, the directory where you're mounting the file system, must exist when you use the `mount` command. If it doesn't exist, create it with `mkdir` as described on page 213.

See page 196 for more information on file systems, mounting, and mount points.

◇ Note that any I/O to the floppy is buffered the same 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 with the command

```
# umount /dev/fd0
```

Do not simply switch floppies as you would on a MS-DOS system. Whenever you change floppies, `umount` the first floppy and `mount` the next.

4.9 Upgrading and installing new software.

Another duty of the system administrator is the upgrading and installation of new software.

Linux system development is rapid. New kernel releases appear every few weeks, and other software is updated nearly 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. This is unnecessary and a waste of time. If you kept up with all of the changes in the Linux world, you would spend all of your time upgrading and none of your time using the system.

Some people feel that you should upgrade when a new distribution release is made; for example, when Slackware comes out with a new version. Many Linux users completely reinstall their system with the newest Slackware release every time.

The best way to upgrade your system depends on the Linux distribution you have. Debian, S.u.S.E., Caldera and Red Hat Linux all have intelligent package management software which allows easy upgrades by installing a new package. For example, the C compiler, `gcc`, comes in a pre-built binary package. When it is installed, all of the files of the older version are overwritten or removed.

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 that a new version offers, then upgrade. If not, don't upgrade. In other words, upgrade only what you must, when you must. Don't upgrade for the sake of upgrading. This wastes a lot of time and effort.

4.9.1 Upgrading the kernel

Upgrading the kernel is a matter of obtaining the kernel sources and compiling them. This is generally a painless procedure, but you can run into problems if you try to upgrade to a development kernel, or upgrade to a new kernel version. The version of a kernel has two parts, the kernel version and patchlevel. As of the time of this writing, the latest stable kernel is version `2.0.33`. The `2.0` is the kernel version and `33` is the patch level. Odd-numbered kernel versions like `2.1` are development kernels. Stay away from development kernels unless you want to live dangerously! As a general rule, you should be able to upgrade easily to another patch level, but upgrading to a new version requires the upgrade of system utilities which interact closely with the kernel.

The Linux kernel sources may be retrieved from any of the Linux FTP sites (see page 310 for a list). On `sunsite.unc.edu`, for instance, the kernel sources are found in `/pub/Linux/kernel`, organized into subdirectories by version number.

Kernel sources are released as a gzipped tar file. For example, the file containing the 2.0.33 kernel sources is `linux-2.0.33.tar.gz`.

Kernel sources are unpacked in the `/usr/src` directory, creating the directory `/usr/src/linux`. It is common practice for `/usr/src/linux` to be a soft link to another directory which contains the version number, like `/usr/src/linux-2.0.33`. This way, you can install new kernel sources and test them out before removing the old kernel sources. The commands to create the kernel directory link are

```
# cd /usr/src
# mkdir linux-2.0.33
# rm -r linux
# ln -s linux-2.0.33 linux
# tar xzf linux-2.0.33.tar.gz
```

When upgrading to a newer patchlevel of the same kernel version, kernel patch files can save file transfer time because the kernel source is around 7MB after being compressed by `gzip`. To upgrade from kernel 2.0.31 to kernel 2.0.33, you would download the patch files `patch-2.0.32.gz` and `patch-2.0.33.gz`, which can be found at the same FTP site as the kernel sources. After you have placed the patches in the `/usr/src` directory,

apply the patches to the kernel in sequence to update the source. One way to do this would be

```
# cd /usr/src
# gzip -cd patch-2.0.32.gz | patch -p0
# gzip -cd patch-2.0.33.gz | patch -p0
```

After the sources are unpacked and any patches have been applied, you need to make sure that three symbolic links in `/usr/include` are correct for your kernel distribution. To create these links use the commands

```
# cd /usr/include
# rm -rf asm linux scsi
# ln -s /usr/src/linux/include/asm-i386 asm
# ln -s /usr/src/linux/include/linux linux
# ln -s /usr/src/linux/include/scsi scsi
```

After you create the links, there is no reason to create them again when you install the next kernel patch or a newer kernel version. (See Section 3.11 for more about symbolic links.)

In order to compile the kernel, you must have the `gcc` C compiler installed on your system. `gcc` version 2.6.3 or a more recent version is required to compile the 2.0 kernel.

First `cd` to `/usr/src/linux`. The command `make config` prompts you for a number of configuration options. This is the step where you select the hardware that your kernel will support. The biggest mistake to avoid is not including support for your hard disk controller. Without the correct hard disk support in the kernel, the system won't even boot. If you are unsure about what a kernel option means, a short description is available by pressing `?` and `Enter`.

Next, run the command `make dep` to update all of the source dependencies. This is an important step. `make clean` removes old binary files from the kernel source tree.

The command `make zImage` compiles the kernel and writes it to `/usr/src/linux/arch/i386/boot/zImage`. Linux kernels on Intel systems are always compressed. Sometimes the kernel you want to compile is too large to be compressed with the compression system that `make zImage` uses. A kernel which is too large will exit the kernel compile with the error message: `Kernel Image Too Large`. If this happens, try the command `make bzImage`, which uses a compression system that supports larger kernels. The kernel is written to `/usr/src/linux/arch/i386/boot/bzImage`.

Once you have the kernel compiled, you need to either copy it to a boot floppy (with a

command like "`cp zImage /dev/fd0`") or install the image so LILO will boot from your hard drive. See page 189 for more information.

4.9.2 Adding a device driver to the kernel.

Page 213 describes how to use an Iomega Zip drive to make backups. Support for the Iomega Zip drive, like many other devices, is not generally compiled into stock Linux distribution kernels—the variety of devices is simply too great to support all of them in a usable kernel. However, the source code for the Zip parallel port device driver is included as part of the kernel source code distribution. This section describes how to add support for an Iomega Zip parallel port drive and have it co-exist with a printer connected to a different parallel port.

You must have installed and sucessfully built a custom Linux kernel, as described in the previous section.

Selecting the Zip drive `ppa` device as a kernel option requires that you answer `Y` to the appropriate questions during the `make config` step, when you determine the configuration of the custom kernel. In particular, the `ppa` device requires answering "`Y`" to three options:

```
SCSI support?  [Y/n/m] Y
SCSI disk support?  [Y/n/m] Y
IOMEGA Parallel Port Zip Drive SCSI support?  [Y/n/m] Y
```

After you have sucessfully run `make config` with all of the support options you want included in the kernel, then run `make dep`, `make clean`, and `make zImage` to build the kernel, you must tell the kernel how to install the driver. This is done via a command line to the LILO boot loader. As described in section 4.2.1, the LILO configuration file, `/etc/lilo.conf` has "stanzas" for each operating system that it knows about, and also directives for presenting these options to the user at boot time.

Another directive that LILO recognizes is "`append=`", which allows you to add boot-time information required by various device drivers to the command line. In this case, the Iomega Zip `ppa` driver requires an unused interrupt and I/O port address. This is exactly analogous to specifying separate printer devices like `LPT1:` and `LPT2:` under MS-DOS.

For example, if your printer uses the hexadecimal (base 16) port address `0x378` (see the installation manual for your parallel port card if you don't know what the address is) and is polled (that is, it doesn't require an IRQ line, a common Linux configuration), you would place the following line in your system's `/etc/lilo.conf` file:

```
append="lp=0x378,0"
```

It is worth noting that Linux automatically recognizes one `/dev/lp` port at boot time, but when specifying a custom port configurations, the boot-time instructions are needed.

The "`0`" after the port address tells the kernel *not* to use a IRQ (interrupt request) line for the printer. This is generally acceptable because printers are much slower than CPUs, so a slower method of accessing I/O devices, known as **polling**, where the kernel periodically checks the printer status on its own, still allows the computer to keep up with the printer.

However, devices that operate at higher speeds, like serial lines and disks, each require an **IRQ,** or **interrupt request,** line. This is a hardware signal sent by the device to the processor whenever the device requires the processor's attention; for example, if the device has data waiting to be input to the processor. The processor stops whatever it is doing and handles the interrupt request of the device. The Zip drive `ppa` device requires a free interrupt, which must correspond to the interrupt that is set on the printer card that you connect the Zip drive to. At the time of this writing, the Linux `ppa` device driver does not support "chaining" of parallel port devices, and separate parallel ports must be used for the Zip `ppa` device and each printer.

To determine which interrupts are already in use on your system, the command

```
# cat /proc/interrupt
```

displays a list of devices and the IRQ lines they use. However, you also need to be careful not to use any automatically configured serial port interrupts as well, which may not be listed in the `/proc/interrupt` file. The Linux Documentation Project's Serial HOWTO, available from the sources listed in Appendix A, describes in detail the configuration of serial ports.

◇ You should also check the hardware settings of various interface cards on your machine by opening the machine's case and visually checking the jumper settings if necessary, to ensure that you are not co-opting an IRQ line that is already in use by another device. Multiple devices fighting for an interrupt line is perhaps the single most common cause of non-functioning Linux systems.

A typical `/proc/interrupt` file looks like

```
0:  6091646 timer
1:  40691 keyboard
2:  0 cascade
4:  284686 + serial
13:  1 math error
14:  192560 + ide0
```

The first column is of interest here. These are the numbers of the IRQ lines that are in use on the system. For the `ppa` driver, we want to choose a line which is *not* listed. IRQ 7 is often a good choice, becuase it is seldom used in default system configurations. We also need to specify the port address which the `ppa` device will use. This address needs to be physically configured on the interface card. Parallel I/O ports are assigned specific addresses, so you will need to read the documentation for your parallel port card. In this example, we will use the I/O port address `0x278`, which corresponds to the LPT2: printer port under MS-DOS. Adding both the IRQ line and port address to our boot-time command line, above, yields the following statement as it would appear in the appropriate stanza of the `/etc/lilo.conf` file:

```
append="lp=0x378,0 ppa=0x278,7"
```

These statements are appended to the kernel's start-up parameters at boot time. They ensure that any printer attached to the system does not interfere with the Zip drive's operation. Of course, if your system does not have a printer installed, the "`lp=`" directive can, and should, be omitted.

After you have installed the custom kernel itself, as described in section 4.2.1, and before you reboot the system, be sure to run the command

```
# /sbin/lilo
```

to install the new LILO configuration on the hard drive's boot sector.

4.9.3 Installing a device driver module.

Page 214 describes how to back up files to a tape drive. Linux provides support for a variety of tape drives with IDE, SCSI, and some proprietary interfaces. Another common type of tape drive connects directly to the floppy drive controller. Linux provides the ftape device driver as a module.

At the time of this writing, the most recent version of ftape is 3.04d. You can retrieve the package from the `sunsite.unc.edu` FTP archive (see Appendix B for instructions). The ftape archive is located in `/pub/Linux/kernel/tapes`. Be sure to get the most recent version. At the time of this writing, this is `ftape-3.04d.tar.gz`.

After unpacking the ftape archive in the `/usr/src` directory, typing `make install` in the top-level ftape directory will compile the ftape driver modules and utilities, if necessary, and install them. If you experience compatibility problems with the ftape executable distribution files and your system kernel or libraries, executing the commands

`make clean` and `make install` will ensure that the modules are compiled on your system.

◇ To use this version of the ftape driver, you must have module support compiled into the kernel, as well as support for the `kerneld` kernel daemon. However, you must *not* include the kernel's built-in ftape code as a kernel option, as the more recent ftape module completely replaces this code.

`make install` also installs the device driver modules in the correct directories. On standard Linux systems, modules are located in the directory

`/lib/modules/`*kernel-version*

If your kernel version is 2.0.30, the modules on your system are located in `/lib/modules/2.0.30`. The `make install` step also insures that these modules are locatable by adding the appropriate statements to the `modules.dep` file, located in the top-level directory of the module files, in this case `/lib/modules/2.0.30`. The ftape installation adds the following modules to your system (using kernel version 2.0.30 in this example):

```
/lib/modules/2.0.30/misc/ftape.o
/lib/modules/2.0.30/misc/zft-compressor.o
/lib/modules/2.0.30/misc/zftape.o
```

The instructions to load the modules also need to be added to the system-wide module configuration file. This is the file `/etc/conf.modules` on many systems. To automatically load the ftape modules on demand, add the following lines to the `/etc/conf.modules` file:

```
alias char-major-27 zftape
pre-install ftape /sbin/swapout 5
```

The first statement loads all of the ftape related modules if necessary when a device with the major number 27 (the ftape device) is accessed by the kernel. Because support for the zftape module (which provides automatic data compression for tape devices) requires the support of the other ftape modules, all of them are loaded on demand by the kernel. The second line specifies load-time parameters for the modules. In this case, the utility `/sbin/swapout`, which is provided with the ftape package, ensures that sufficient DMA memory is available for the ftape driver to function.

To access the ftape device, you must first place a formatted tape in the drive. Instructions for formatting tapes and operation of the tape drive are given in section 4.8.3.

4.9.4 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 backwardly compatible. 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 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 tar files compressed with `gzip`. 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 with each release). In general, you need to install the library's `.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 run time by programs using the libraries. Each library has a symbolic link using the major version number of the library in `/lib`.

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 placed in `/lib` pointing to the library. You must 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.

◇ You must change the symbolic link in one step, as described below. If you 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 the appropriate file names,

above. The library release notes should explain the details. (See page 164 for more information about symbolic links.)

4.9.5 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. Most distributions have upgrade versions that work with their package management software. In general, these packages are much easier to install than "generic" distributions.

4.9.6 Upgrading other software.

Upgrading other software is often simply a matter of downloading the appropriate files and installing them. Most software for Linux is distributed as compressed tar files that include sources, binaries, or both. If binaries are not included in the release, you may need to compile them yourself. This means at least typing `make` in the directory where the sources are located.

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 you want is the easiest way to locate software. If you have `archie` available to you, it can be of assistance as well[3]. There are also other Internet resources which are devoted specifically to Linux. See Appendix A for more details.

4.10 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.

[3] 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`"

4.10.1 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 file systems (see page 196) and initializing swap space. On some systems, `init` executes the file `/etc/init.d/rc`.

Your system may execute other initialization scripts as well. For example `/etc/rc.local` which usually contains initialization commands specific to your own system, such as setting the host name (see the next section). `rc.local` may be started from `/etc/rc` or from `/etc/inittab` directly.

4.10.2 Setting the host name.

In a networked environment, the host name is used to uniquely identify a particular machine, while on a standalone machine, the host name simply 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 host name is a simple matter of using the `hostname` command. If you are on a network, your host name should be the full host name 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 domain name, such as `loomer.vpizza.com`, `shoop.nowhere.edu`, or `floof.org`.

The host name 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 host name in `/etc/hosts`. If you are not on a TCP/IP network, and your host name is `floof.org`, simply include the following line in `/etc/hosts`:

```
127.0.0.1       floof.org localhost
```

This assigns your host name, `floof.org`, to the loopback address 127.0.0.1. The loopback interface is present whether the machine is connected to a network or not. The `localhost` alias is always assigned to this address.

If you are on a TCP/IP network, your actual IP address and host name should appear in `/etc/hosts`. For example, if your host name 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
```

To set your host name, simply use the `hostname` command. For example, the command

```
# hostname -S goober.norelco.com
```

sets the host name to `goober.norelco.com`. In most cases, the `hostname` command is executed from one of the system startup files, like `/etc/rc` or `/etc/rc.local`. Edit these two files and change the `hostname` command found there to set your own host name. Upon rebooting, the system will use the new name.

4.11 What to do in an emergency.

On some occasions, the system administrator will be faced with the problem of recovering from a complete disaster, such as forgetting the root password or trashing file systems. 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 than with commercial versions of UNIX on many platforms. Linux also benefits from a strong complement of wizards who can help you out of a bind. (The *double entendre* is intended.)

The first step to fixing any problem youself is finding out what it is. Poke around, and see how things work. Much of the time, a system administrator posts a desperate plea for help before he or she looks into the problem at all. You'll find that fixing problems yourself is actually very easy. It is the path to enlighenment and guruhood.

There are a few times when 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 for help. In many cases, you can recover your system from a maintenance diskette.

4.11.1 Recovery with a maintenance diskette.

One indispensable tool of the system administrator is the so-called "boot/root disk," a floppy that can be booted for a complete Linux system, independent of your hard drive. Boot/root disks are actually very simple—you create a root file system 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 file system. In any case, the result is the same: you are running a Linux system completely from the floppy drive.

The canonical example of a boot/root disk is the Slackware boot disks. These diskettes contain a bootable kernel and a root file system, all on floppy. They are intended to be used to install the Slackware distribution, but come in handy when doing system maintenance.

The H.J Lu boot/root disk, available from `/pub/Linux/GCC/rootdisk` on `sunsite.unc.edu`, is another example of a maintenance disk. If you're ambitious, you can create your own. In most cases, however, a ready-made boot/root disk is much easier to use and probably will be more complete.

Using a boot/root disk is very simple. Boot the disk on your system, and login as `root` (usually with no password). In order to access the files on the hard drive, you will need to mount the file systems by hand. For example, the command

```
# mount -t ext2 /dev/hda2 /mnt
```

will mount an ext2fs file system on `/dev/hda2` under `/mnt`. Remember that `/` is now on the boot/root disk itself; you need to mount your hard drive file systems 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 file system on `/mnt`.

4.11.2 Fixing the root password.

If you forget your root password, this is not a problem, surprisingly. Boot the boot/root disk, mount the root file system on `/mnt`, and blank out the password field for `root` in `/mnt/etc/passwd`, as in this example:

```
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 that you learned how to use `vi`? On your boot/root disk, editors like Emacs probably aren't available, but `vi` should be.

4.11.3 Trashed file systems.

If you somehow trash a file systems, you can run `e2fsck` or the appropriate form of `fsck` for the file system type. (See page 199.) In most cases, it is safest to correct any damaged data on the hard drive file systems from floppy.

One common cause of file system damage is a damaged super block. The **super block** is the "header" of the file system that contains information about its status, size, free blocks, and so forth. If you damage the super block, by accidentally writing data directly to the file system's partition table for example, the system probably will not recognize the file system at all. Attempt to mount the file system will fail, and `e2fsck` won't be able to fix the problem.

Happily, an ext2fs file system type saves copies of the superblock at "block group" boundaries on the drive, usually every 8K blocks. To tell `e2fsck` to use a copy of the superblock, use a command like

```
# e2fsck -b 8193 partition
```

where *partition* is the partition on which the file system resides. The `-b 8193` option tells `e2fsck` to use the copy of the superblock stored at block 8193 in the file system.

4.11.4 Recovering lost files.

If you accidentally delete an important file on your system, there's no way to "undelete" it. However, you can copy the relevant files from the floppy to your hard drive. If you delete `/bin/login`, for example, which allows you to login, simply boot the boot/root floppy, mount the root file system 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 aren't essential system files that have counterparts on the boot/root floppy, you're out of luck. If you make backups however, you can always restore them.

4.11.5 Trashed libraries.

If you accidentally trash your libraries or symbolic links in `/lib`, more than likely the commands which depend on the libraries will no longer work (see page 223). The easiest solution is to boot your boot/root floppy, mount your root file system, and fix the libraries in `/mnt/lib`. Page 223 describes how install run time libraries and their symbolic links.

Chapter 5

The X Window System

The X Window System is a graphical user interface (GUI) that was originally developed at the Massachusetts Institute of Technology. Commercial vendors have since made X the industry standard GUI for UNIX platforms. Virtually every UNIX workstation in the world now runs some form of X.

A free port of the MIT X Window System version 11, release 6 (X11R6) for 80386, 80486, and Pentium UNIX systems was developed by a team of programmers that was originally headed by David Wexelblat. This release, known as XFree86[1], is available for System V/386, 386BSD, and other Intel x86 UNIX implementations, including Linux. It provides all of the binaries, support files, libraries, and tools required for installation.

Some features offered by this release are:

- complete inclusion of the X Consortium's X11R6.3 release;
- a new DPMS extension, donated by Digital Equipment Corporation;
- the Low Bandwidth X (LBX) extension in all X servers;
- Microsoft IntelliMouse support;
- support for `gzip` font compression.

To use the X Window System, you are encouraged to read *The X Window System: A User's Guide* (see Appendix A). Here, we describe step-by-step an XFree86 installation

[1] XFree86 is a trademark of The XFree86 Project, Inc.

under Linux. You still need to fill in some of the details by reading the XFree86 documentation, which is discussed below. The Linux *XFree86 HOW TO* is another good information source.

5.1 X Window Hardware requirements.

5.1.1 Video display.

The documentation for your video adaptor should specify the chip set. If you are in the market for a new video card, or are buying a machine that comes with a video card, ask the vendor to find out exactly what make, model, and chip set the video card comes with. The vendor may need to call the manufacturer's technical support department. Many personal computer hardware vendors state their video card is a "standard SVGA card," that "should work," with your system. Explain that your software (mention Linux and XFree86!) does not support all video chip sets, and that you must have detailed information.

You can also determine your video card chip set by running the `SuperProbe` program which is included with the XFree86 distribution. This is described below.

Video cards using these chip sets are supported on all bus types. Virtually all of the cards support 256-color graphics modes. In addition, some of the cards support color modes like monochrome, 15-bit, 16-bit, 24-bit and 32-bit. For color depths greater than 256 (8-bit), you must have the requisite amount of video dynamic RAM (DRAM) installed. The usual configuration is 16 bits per pixel (65536 colors).

The monochrome server also supports generic VGA cards, the Hercules monochrome card, the Hyundai HGC1280, Sigma LaserView, and Apollo monochrome cards.

The release notes for the current version of XFree86 should contain the complete list of supported video chip sets. The XFree86 distribution has chip set-specific README files that give detailed information on the state of support for each chip set.

One problem faced by the XFree86 developers is that some video card manufacturers use non-standard mechanisms to determine the clock frequencies that are used to drive their card. They either don't release specifications which describe how to program the card or require developers to sign non-disclosure statements to get the information. This practice restricts the free distribution of XFree86, and the XFree86 development team is unwilling to accept it. This has been a problem with older video cards manufactured by Diamond, but as of release 3.3, Diamond actively supports the XFree86 Project, Inc.

We also suggest using an accelerated card, like a S3 chip set card. You should check the XFree86 documentation and verify that your particular card is supported

before you take the plunge and purchase expensive hardware. Benchmark comparisons of video cards under XFree86 are posted routinely to the Usenet news groups `comp.windows.x.i386unix` and `comp.os.linux.misc`.

It is important to note that the average accelerated video card is significantly faster than the standard graphics card of most workstations. An 80486DX2, 66-MHz Linux System with 20 megabytes of RAM, equipped with a VESA Local Bus (VLB) S3-864 chip set card with 2 megabytes of DRAM, will consistently be about 7 times as fast a Sun Sparc IPX workstation on X benchmarks with the XFree86 server version 3.1. Version 3.3 is even faster. In general, a Linux system with an accelerated SVGA card will give you much greater performance than commercial UNIX workstations, which usually employ simple frame buffers for graphics.

5.1.2 Memory, CPU, and disk space.

The suggested setup for XFree86 under Linux is a 80486 or faster machine with at least 16 megabytes of RAM. The more physical RAM installed, the less the system must swap to and from the disk when memory is low. Because swapping is inherently slow (disks are very slow compared to memory), having 16 megabytes of RAM or more is necessary to run XFree86 comfortably. A system with 4 megabytes of physical RAM could run 10 to 100 times more slowly than one with 16 megabytes or more.

A standartd, out-of-the-box XFree86 installation requires 60–80 megabytes of disk space, at a minimum. This includes space for the X server(s), fonts, libraries, and standard utilities. If you plan to add applications, you can probably run XFree86 comfortably in 200 megabytes of disk space.

5.2 XFree86 installation.

The Linux binary distribution of XFree86 is found on all CD Linux distributions and can also be found at a number of FTP sites. On `sunsite.unc.edu`, it is found in the directory `/pub/X11/XFree86`. At of the time of this writing, the current version is 3.3.1. Newer versions are released periodically. If you obtain XFree86 as part of a Linux distribution, downloading the software separately is not necessary.

These files are included in the XFree86-3.3.1 distribution.

One of the following servers is required:

File	Description
`X338514.tgz`	Server for 8514-based boards.
`X33AGX.tgz`	Server for AGX-based boards.
`X33I128.tgz`	Server for the Imagine I128 boards.
`X33Ma64.tgz`	Server for Mach64-based boards.
`X33Ma32.tgz`	Server for Mach32-based boards.
`X33Ma8.tgz`	Server for Mach8-based boards.
`X33Mono.tgz`	Server for monochrome video modes.
`X33P9K.tgz`	Server for P9000-based boards.
`X33S3.tgz`	Server for S3-based boards.
`X33S3V.tgz`	Server for S3/Virge-based boards.
`X33SVGA.tgz`	Server for Super VGA-based boards.
`X33VGA16.tgz`	Server for VGA/EGA-based boards.
`X33W32.tgz`	Server for ET4000/W32-based boards.

All of the following files are required:

File	Description
`preinst.sh`	Pre-installation script
`postinst.sh`	Post-installation script
`X33bin.tgz`	Clients, run-time libs, and app-defaults files
`X33doc.tgz`	Documentation
`X33fnts.tgz`	75dpi, misc and PEX fonts
`X33lib.tgz`	Data files required at run-time
`X33man.tgz`	Manual pages
`X33set.tgz`	XF86Setup utility
`X33VG16.tgz`	16 colour VGA server (XF86Setup needs this server)

The following is required for new installations, and optional for existing installations:

File	Description
`X33cfg.tgz`	sample config files for xinit, xdm

◇ Do not install `X33cfg.tgz` over an existing XFree86 installation without first backing up the configuration files. Unpacking `X33cfg.tgz` overwrites these and other files. If you do have customized configuration files, there is no need to install this package anyway.

◇ The bit mapped fonts distributed with release 3.3.1 are compressed with the `gzip` program rather than `compress`. You will probably want to remove the old fonts after you back them up. The X servers and font servers in previous releases cannot read fonts compressed by `gzip`, so keep a copy of the old fonts if you want to use older servers.

The following files are optional:

File	Description
`X33f100.tgz`	100dpi fonts
`X33fcyr.tgz`	Cyrillic fonts
`X33fnon.tgz`	Other fonts (Chinese, Japanese, Korean, Hebrew)
`X33fscl.tgz`	Scalable fonts (Speedo and Type1)
`X33fsrv.tgz`	Font server and config files
`X33prog.tgz`	X header files, config files and compile-time libs
`X33nest.tgz`	Nested X server
`X33vfb.tgz`	Virtual framebuffer X server
`X33prt.tgz`	X Print server
`X33ps.tgz`	PostScript version of the documentation
`X33html.tgz`	HTML version of the documentation
`X33jdoc.tgz`	Documentation in Japanese (for version 3.2)
`X33jhtm.tgz`	HTML version of the documentation in Japanese (3.2)
`X33lkit.tgz`	X server LinkKit

The XFree86 directory should contain `README` files and installation notes for the current version.

Next, as root, create the directory `/usr/X11R6` if it doesn't already exist. Then run the pre-installation script, `preinst.sh`. You should copy the script, and all of the archive files for your system to the `/var/tmp` directory before you run `preinst.sh`. `/usr/X11R6` must be your current directory when you run the pre-installation script and unpack the archives.

```
# cd /usr/X11R6
# sh /var/tmp/preinst.sh
```

You should then unpack the files from `/var/tmp` to `/usr/X11R6` with a command like:

```
# gzip -d < /var/tmp/X33prog.tgz | tar vxf -
```

◇ These `tar` files are packed relative to `/usr/X11R6`. You must unpack the files there. On some Linux distributions, the parent directory is `/var/X11R6` instead.

After you have upacked the required files and any optional files you selected, run the post-installation script `postinst.sh`.

```
# cd /usr/X11R6
# sh /var/tmp/postinst.sh
```

Now link the file `/usr/X11R6/bin/X` to the server that supports your video card. For example, the SVGA color server, `/usr/bin/X11/X` should be linked to `/usr/X11R6/bin/XF86_SVGA`. To use the monochrome server instead, relink `X` to `XF86_MONO` with the command

```
# ln --sf /usr/X11R6/bin/XF86_MONO  /usr/X11R6/bin/X
```

The same holds true for the other servers.

You also need to ensure that the directory, `/usr/X11R6/bin`, is on your path. This can be done by editing your system default `/etc/profile` or `/etc/csh.login` (based on the shell that you, or other users on your system, use). Or you can simply add the directory to your personal path by modifying `/etc/.bashrc` or `/etc/.cshrc`, based on your shell.

Finally, ensure that `/usr/X11R6/lib` can be located by `ld.so`, the run time linker. To do this, add the line

```
/usr/X11R6/lib
```

to the file `/etc/ld.so.conf`, and run `/sbin/ldconfig`, as `root`.

5.3 Probing the hardware configuration.

If you aren't sure which server to use or don't know the video card's chip set, the `SuperProbe` program, which is found in `/usr/X11R6/bin`, can attempt to determine the video chip set and other information. Write down its output for later reference.

To run SuperProbe from the command line, simply enter

```
# SuperProbe
```

◇ It is possible that `SuperProbe` will confuse hardware that uses I/O port addresses that might be used by video cards. To prevent SuperProbe from checking these addresses, use the `excl` argument, followed by a list of addresses that `SuperProbe` should not examine. For example:

```
# SuperProbe -excl 0x200-0x230,0x240
```

The addresses are given as hexadecimal numbers that are prefixed by `0x`.

To display a list of video devices that SuperProbe knows about, use the command

```
# SuperProbe -info
```

SuperProbe can print lots of information if you provide it with the -verbose argument. You can redirect the output to a file:

```
# SuperProbe -verbose >superprobe.out
```

◇ Running SuperProbe can cause the system to hang. Make certain that any non-essential applications are not running, or at least have all of their data safely saved to disk, and ensure that any users are logged off. Also, a loaded system (one that is printing in the background, for example), can skew the output of software like SuperProbe or an X server that is trying to measure the video card's timing specifications.

5.4 Automatically generating the XF86Config file.

Creating the XF86Config file by hand is an arduous task, but it is not impossible. Several tools in the XFree86 version 3.3.1 can assist you. One of them, the XF86Setup program, can automatically generate an XF86Config file in the correct format. You must know the exact specifications of your video board and the Vertical and Horizontal refresh values of your monitor. Most of the information can be found in the owner's manuals.

Several other configuration programs are available as well, depending on the Linux distribution. The most common ones are Xconfigurator and xf86config. The latter program is an older version of XF86Setup and is included in older releases of XFree86. You should always use XF86Setup if both it and xf86config are available.

5.5 Configuring XFree86.

In this section, we describe how to create and edit the XF86Config file, which configures the XFree86 server. In many cases, it is best to start with an XFree86 configuration that uses a low resolution like 640x480 that is supported by nearly all video cards and monitors. Once XFree86 works at a lower, standard resolution, you can tweak the configuration to exploit the capabilities of your video hardware. This ensures that XFree86 works on your system, and that the installation is essentailly correct, before you attempt the sometimes difficult task of setting up XFree86 for high-performance use.

In addition to the information listed here, you should read the following documents:

- The XFree86 documentation in /usr/X11R6/lib/X11/doc (from the XFree86-3.1-doc package). You should especially see the file README.Config, which is an XFree86 configuration tutorial.

- Several video chip sets have separate `README` files in the above directory (like `README.Cirrus` and `README.S3`). Read the file that applies to your video card.
- The manual page for `XFree86`.
- The manual page for `XF86Config`.
- The manual page for the server that you are using, like `XF86_SVGA` or `XF86_S3`.

The main XFree86 configuration file is `/usr/X11R6/lib/X11/XF86Config`. This file contains information for your mouse, video card parameters, and so on. The file `XF86Config.eg` is provided with the XFree86 distribution as an example. Copy this file to `XF86Config` and edit it as a starting point.

The `XF86Config` manual page explains the format of the `XF86Config` file. Read the manual page if you have not done so already.

We are going to describe a sample `XF86Config` file, a section at a time. This file may not look exactly like the sample file included in the XFree86 distribution, but the structure is the same.

◇ Note that the `XF86Config` file format may change with each version of XFree86. See your distribution's release notes for errata.

◇ **Do not copy the configuration file listed here to your system and try to use it.** A configuration file that does not correspond to your hardware can drive the monitor at a frequency which is too high. There have been reports of damage to monitors, especially fixed-frequency monitors, that has been caused by incorrectly configured `XF86Config` files. **Make absolutely sure that your `XF86Config` file corresponds to your hardware before you use it.**

Each section of the `XF86Config` file is surrounded by a pair of lines with the syntax, `Section "`*section-name*`"...EndSection`.

The first section of the `XF86Config` file is `Files`, which looks like this:

```
Section "Files"
    RgbPath     "/usr/X11R6/lib/X11/rgb"
    FontPath    "/usr/X11R6/lib/X11/fonts/misc/"
    FontPath    "/usr/X11R6/lib/X11/fonts/75dpi/"
EndSection
```

The `RgbPath` line sets the path to the X11R6 RGB color database, and each `FontPath` line sets the path to a directory containing X11 fonts. You shouldn't have to modify these lines. Simply ensure that a `FontPath` entry exists for each font type that you have installed; that is, for each directory in `/usr/X11R6/lib/X11/fonts`.

The next section is `ServerFlags`, which specifies several global flags for the server. In general this section is empty.

```
Section "ServerFlags"
# Uncomment this to cause a core dump at the spot where a signal is
# received. This may leave the console in an unusable state, but may
# provide a better stack trace in the core dump to aid in debugging
#    NoTrapSignals

# Uncomment this to disable the <Crtl><Alt><BS> server abort sequence
#    DontZap
EndSection
```

In this `ServerFlags` section, all of the lines are commented out.

The next section is `Keyboard`. This example shows a basic configuration that should work on most systems. The `XF86Config` file describes how to modify the configuration.

```
Section "Keyboard"
    Protocol    "Standard"
    AutoRepeat  500 5
    ServerNumLock
EndSection
```

The next section is `Pointer`, which specifies parameters for the mouse device:

```
Section "Pointer"

    Protocol    "MouseSystems"
    Device      "/dev/mouse"

# Baudrate and SampleRate are only for some Logitech mice
#    BaudRate   9600
#    SampleRate 150

# Emulate3Buttons is an option for 2-button Microsoft mice
#    Emulate3Buttons

# ChordMiddle is an option for some 3-button Logitech mice
#    ChordMiddle

EndSection
```

For the moment, the only options that should concern you are `Protocol` and `Device`. `Protocol` specifies the mouse *protocol*, which is not necessarily the same as the manufacturer. XFree86 under Linux recognizes these mouse protocols:

- `BusMouse`
- `Logitech`
- `Microsoft`
- `MMSeries`
- `Mouseman`
- `MouseSystems`
- `PS/2`
- `MMHitTab`

`BusMouse` should be used for the Logitech bus mice. Older Logitech mice use `Logitech`, and newer Logitech serial mice use either `Microsoft` or `Mouseman` protocols.

`Device` specifies the device file by which the mouse can be accessed. On most Linux systems, this is `/dev/mouse`, which is usually a link to the appropriate serial port, like `/dev/cua0` for serial mice and the appropriate bus mouse device for bus mice. At any rate, be sure that the device file exists.

The next section is `Monitor`, which specifies the characteristics of your monitor. As with other sections in the `XF86Config` file, there may be more than one `Monitor` section. This is useful if you have multiple monitors connected to a system, or use the same `XF86Config` file for multiple hardware configurations.

```
Section "Monitor"

    Identifier  "CTX 5468 NI"

    # These values are for a CTX 5468NI only! Don't attempt to use
    # them with your monitor (unless you have this model)

    Bandwidth    60
    HorizSync    30-38,47-50
```

```
    VertRefresh  50-90

    # Modes: Name      dotclock  horiz                vert

    ModeLine "640x480"  25       640 664 760 800      480 491 493 525
    ModeLine "800x600"  36       800 824 896 1024     600 601 603 625
    ModeLine "1024x768" 65       1024 1088 1200 1328  768 783 789 818

EndSection
```

`Identifier` is an arbitrary name for the `Monitor` entry. This can be any string and is used to refer to the `Monitor` entry later in the `XF86Config` file.

`HorizSync` specifies the valid horizontal sync frequencies for your monitor, in kHz. Multisync monitors may have a range of values, or several, comma separated ranges. Fixed-frequency monitors require a list of discrete values; for example:

```
    HorizSync    31.5, 35.2, 37.9, 35.5, 48.95
```

The monitor manual should list these values in the technical specifications section. If it does not, contact the manufacturer or vendor of your monitor to obtain it.

`VertRefresh` specifies the valid vertical refresh rates (or vertical synchronization frequencies) for your monitor, in kHz. Like `HorizSync`, this can be a range or a list of discrete values. Your monitor manual should list them.

`HorizSync` and `VertRefresh` are used only to double-check that the monitor resolutions are in valid ranges. This reduces the chance that you will damage your monitor by driving it at a frequency which it was not designed for.

The `ModeLine` directive is used to specify resolution modes for your monitor. The format is

`ModeLine` *name clock horiz-values vert-values*

name is an arbitrary string which you will use to refer to the resolution mode later in the file. *dot-clock* is the driving clock frequency, or "dot clock" associated with the resolution mode. A dot clock is usually specified in MHz. It is the rate at which the video card must send pixels to the monitor at this resolution. *horiz-values* and *vert-values* are four numbers each that specify when the electron gun of the monitor should fire, and when the horizontal and vertical sync pulses fire during a sweep.

The file `VideoModes.doc`, included with the XFree86 distribution, describes in detail how to determine the `ModeLine` values for each resolution mode that your monitor

supports. *clock* must correspond to one of the dot clock values that your video card supports. Later in the `XF86Config` file, you will specify these clocks.

Two files, `modeDB.txt` and `Monitors`, may have `ModeLine` information for your monitor. They are located in `/usr/X11R6/lib/X11/doc`.

Start with `ModeLine` values for VESA-standard monitor timings, because most monitors support them. `ModeDB.txt` includes the timing values for VESA-standard resolutions. For example, this entry,

```
# 640x480@60Hz Non-Interlaced mode
# Horizontal Sync = 31.5kHz
# Timing: H=(0.95us, 3.81us, 1.59us), V=(0.35ms, 0.064ms, 1.02ms)
#
# name        clock   horizontal timing     vertical timing      flags
 "640x480"     25.175  640  664  760  800    480  491  493  525
```

is the VESA-standard timing for a 640x480 video mode. It has a dot clock of 25.175, which your video card must support. This is described below. To include this entry in the `XF86Config` file, use the line

```
ModeLine "640x480" 25.175  640 664 760 800  480 491 493
525
```

The *name* argument to `ModeLine` (`"640x480"`) is an arbitrary string. By convention modes are named by their resolutions, but *name* can, technically, be any descriptive label.

For each `ModeLine`, the server checks the mode specifications and ensures that they fall in the range of values specified for `Bandwidth`, `HorizSync`, and `VertRefresh`. If they do not, the server complains when you attempt to start X. For one thing, the dot clock used by the mode should not be greater than the value used for `Bandwidth`. However, in many cases, it is safe to use a mode that has a slightly higher bandwidth than your monitor can support.

If the VESA standard timings do not work, (you'll know after you try to use them), then look in the files `modeDB.txt` and `Monitors`, which include specific mode values for many monitor types. You can create `ModeLine` entries from these values as well. Be sure only to use values for your specific monitor. Many 14 and 15-inch monitors do not support higher resolution modes, and often resolutions of 1024x768 at low dot clocks. If you can't find high-resolution modes for your monitor in these files, then your monitor probably does not support them.

If you are completely at a loss and can't find `ModeLine` values for your monitor, follow the instructions in the `VideoModes.doc` file, which is included in the XFree86

distribution, and generate values from the specifications in your monitor's manual. Your mileage will certainly vary when you attempt to generate `ModeLine` values by hand. But this is a good place to look if you can't find the values that you need. `VideoModes.doc` also describes the format of the `ModeLine` directive, and other aspects of the XFree86 server in gory detail.

Lastly, if you do obtain `ModeLine` values that are almost but not exactly right, you may possibly be able to modify the values a little to obtain the desired result. For example, if the XFree86 display image is shifted slightly, or the image seems to "roll," then follow the instructions in the `VideoModes.doc` file and fix the values. Be sure to check the controls on the monitor itself. In many cases, you must change the horizontal or vertical size of the display after XFree86 starts, to center and size the image.

◇ Don't use monitor timing values or `ModeLine` values for monitors other than your model. If you try to drive a monitor at a frequency for which it was not designed, you can damage or even destroy it.

The next section of the `XF86Config` file is `Device`, which specifies parameters for your video card. Here is an example.

```
Section "Device"
        Identifier "#9 GXE 64"

        # Nothing yet; we fill in these values later.

EndSection
```

This section defines properties for a particular video card. `Identifier` is an arbitrary, descriptive string. You will use this string to refer to the card later.

Initially, you don't need to include anything in the `Device` section except the `Identifier`. We will use the X server itself to probe for the properties of the video card and enter them into the `Device` section later. The XFree86 server is capable of probing for the video chip set, clocks, RAMDAC, and amount of video RAM on the board. This is described in Section 5.6.

Before we do this, however, we need to finish writing the `XF86Config` file. The next section is `Screen`, which specifies the monitor/video card combination to use for a particular server.

```
Section "Screen"
    Driver     "Accel"
    Device     "#9 GXE 64"
    Monitor    "CTX 5468 NI"
```

```
        Subsection "Display"
            Depth      16
            Modes      "1024x768" "800x600" "640x480"
            ViewPort   0 0
            Virtual    1024 768
        EndSubsection
    EndSection
```

The `Driver` line specifies the X server that you will be using. Valid `Driver` values are:

- `Accel`: For the `XF86_S3`, `XF86_Mach32`, `XF86_Mach8`, `XF86_8514`, `XF86_P9000`, `XF86_AGX`, and `XF86_W32` servers;
- `SVGA`: For the `XF86_SVGA` server;
- `VGA16`: For the `XF86_VGA16` server;
- `VGA2`: For the `XF86_Mono` server;
- `Mono`: For the non-VGA monochrome drivers in the `XF86_Mono` and `XF86_VGA16` servers.

Be sure that `/usr/X11R6/bin/X` is a symbolic link to this server.

The `Device` line specifies the `Identifier` of the `Device` section that corresponds to the video card to use for this server. Above, we created a `Device` section with the line

```
Identifier "#9 GXE 64"
```

Therefore, we use `"#9 GXE 64"` on the `Device` line here.

Similarly, the `Monitor` line specifies the name of the `Monitor` section to be used with this server. Here, `"CTX 5468 NI"` is the `Identifier` used in the `Monitor` section described above.

`Subsection "Display"` defines several properties of the XFree86 server corresponding to your monitor/video card combination. The `XF86Config` file describes all of these options in detail. Most of them are not necessary to get the system working.

The options that you should know about are:

- `Depth`. Defines the number of color planes; that is, the number of bits per pixel. Usually, `Depth` is set to 16. For the `VGA16` server, you would use a depth of 4, and for the monochrome server a depth of 1. If you use an accelerated video card with

enough memory to support more bits per pixel, you can set `Depth` to 24, or 32. If you have problems with depths higher than 16, set it back to 16 and attempt to debug the problem later.

- `Modes`. This is the list of mode names which have been defined using the `ModeLine` directive(s) in the `Monitor` section. In the above section, we used `ModeLine`s named `"1024x768"`, `"800x600"`, and `"640x48"0`. Therefore, we use a `Modes` line of

  ```
  Modes    "1024x768" "800x600" "640x480"
  ```

  The first mode listed on this line is the default when XFree86 starts. After XFree86 is running, you can switch between the modes listed here using the keys `Ctrl`-`Alt`-`Numeric +` and `Ctrl`-`Alt`-`Numeric -`.

  It might be best, when you initially configure XFree86, to use lower resolution video modes like 640x480, which tend to work with most systems. Once you have the basic configuration working, you can modify `XF86Config` to support higher resolutions.

- `Virtual`. Set the virtual desktop size. XFree86 can use additional memory on your video card to extend the size of the desktop. When you move the mouse pointer to the edge of the display, the desktop scrolls, bringing the additional space into view. Even if you run the server at a lower video resolution like 800x600, you can set `Virtual` to the total resolution that your video card can support. A 1-megabyte video card can support 1024x768 at a depth of 8 bits per pixel; a 2-megabyte card 1280x1024 at depth 8, or 1024x768 at depth 16. Of course, the entire area will not be visible at once, but it can still be used.

  The `Virtual` feature is rather limited. If you want to use a true virtual desktop, `fvwm` and similar window managers allow you to have large, virtual desktops by hiding windows and using other techniques, instead of storing the entire desktop in video memory. See the manual pages for `fvwm` for more details about this. Many Linux systems use `fvwm` by default.

- `ViewPort`. If you are using the `Virtual` option which is described above, `ViewPort` sets the coordinates of the upper-left-hand corner of the virtual desktop when XFree86 starts up. `Virtual 0 0` is often used. If this is unspecified, then the desktop is centered on the virtual desktop display, which may be undesirable to you.

Many other options for this section exist; see the `XF86Config` manual page for a complete description. In practice, these options are not necessary to get XFree86 working initially.

5.6 Filling in video card information.

Your `XF86Config` file is now ready, with the exception of complete information on the video card. We'll use the X server to probe for this information, and add it to `XF86Config`.

Instead of probing for this information with the X server, `XF86Config` values for many cards are listed in the files `modeDB.txt`, `AccelCards`, and `Devices`. These files are all found in `/usr/X11R6/lib/X11/doc`. In addition, there are various `README` files for certain chip sets. You should look at these files for information on your video card and use that information (the clock values, chip set type, and any options) in the `XF86Config` file. If any information is missing, you can probe for it.

In most of these examples we demonstrate configuration of a #9 GXE 64 video card, which uses the `XF86_S3` chipset. First, determine the video chip set on the card. Running `SuperProbe` (found in `/usr/X11R6/bin`) tells you this information, but you need to know the chip set name as it is known to the X server.

To do this, run the command

```
X -showconfig
```

This gives the chip set names known to the X server. (The manual pages for each X server list these, too.) For example, with the accelerated `XF86_S3` server, we get:

```
XFree86 Version 3.1 / X Window System
(protocol Version 11, revision 0, vendor release 6000)
Operating System: Linux
Configured drivers:
  S3: accelerated server for S3 graphics adaptors (Patchlevel 0)
      mmio_928, s3_generic
```

The valid chip set names for this server are `mmio_928` and `s3_generic`. The `XF86_S3` man page describes these chip sets and video cards that use them. In the case of the #9 GXE 64 video card, `mmio_928` is appropriate.

If you don't know which chip set is in use, the X server can probe it for you. To do this, run the command

```
X -probeonly > /tmp/x.out 2>&1
```

if you use `bash` as your shell. If you use `csh`, try:

```
X -probeonly &> /tmp/x.out
```

You should run this command while the system is unloaded; that is, while no other activity occurs on the system. This command also probes for your video card dot clocks (as seen below), and system load can throw off this calculation.

The output from the above, in `/tmp/x.out`, should contain lines like:

```
XFree86 Version 3.1 / X Window System
(protocol Version 11, revision 0, vendor release 6000)
Operating System: Linux
Configured drivers:
  S3: accelerated server for S3 graphics adaptors (Patchlevel 0)
      mmio_928, s3_generic
```

Several lines deleted...

```
(--) S3: card type: 386/486 localbus
(--) S3: chipset:   864 rev. 0
(--) S3: chipset driver: mmio_928
```

Here, we see that the two valid chip sets for this server (in this case, `XF86_S3`) are `mmio_928` and `s3_generic`. The server probed for and found a video card that has the `mmio_928` chipset.

In the `Device` section of the `XF86Config` file, add a `Chipset` line that has the name of the chip set as determined above. For example,

```
Section "Device"
        # We already had Identifier here...
        Identifier "#9 GXE 64"
        # Add this line:
        Chipset "mmio_928"
EndSection
```

Now, we need to determine the driving clock frequencies used by the video card. A driving clock frequency, or dot clock, is simply a rate at which the video card can send pixels to the monitor. As described above, each monitor resolution has a dot clock associated with it. We need to determine which dot clocks are made available by the video card.

First, you should look at the documentation mentioned above and see if the card's clocks are listed there. The dot clocks are usually a list of 8 or 16 values, all of which are

in MHz. For example, when looking at `modeDB.txt`, we see an entry for the Cardinal ET4000 video card, which looks like:

```
# chip    ram   virtual   clocks                          default-mode  flags
 ET4000   1024  1024 768   25  28  38  36  40  45  32   0  "1024x768"
```

The dot clocks for this card are 25, 28, 38, 36, 40, 45, 32, and 0 MHz.

In the `Devices` section of the `XF86Config` file, add a `Clocks` line containing the list of dot clocks for your card. For example, for the clocks above, add the line

```
Clocks 25 28 38 36 40 45 32 0
```

to the `Devices` section of the file, after `Chipset`.

◇ **The order of the dot clocks is important.** Don't re-sort the list or remove duplicates.

If you cannot find the dot clocks associated with your card, the X server can probe for these, too. Use `X -probeonly` as described above. The output should contain lines which look like the following:

```
(--) S3: clocks:  25.18  28.32  38.02  36.15  40.33  45.32  32.00  00.00
```

We can then add a `Clocks` line which contains all of these values, as printed. You can use more than one `Clocks` line in `XF86Config` if all of the values (sometimes there are more than 8 clock values printed) do not fit onto one line. Again, be sure to keep the list of clocks in the order that they are displayed.

◇ Be sure that there is no `Clocks` line (or that it is commented out) in the `Devices` section of the file when using `X -probeonly`. If there is a `Clocks` line present, the server does not probe for the clocks—it uses the values given in `XF86Config`.

Some video boards use a programmable clock chip. See the manual page for your X server or the XFree86 `README` file that describes your video card. The chip essentially allows the X server to tell the card the dot clocks to use. For video cards that have clock chips, you may not find a list of dot clocks for the card in any of the above files. Or, the list of dot clocks printed when using `X -probeonly` will only contain one or two discrete clock values, with the rest being duplicates or zero. Or, the X server may provide an explicit warning that the video card has a programmable clock chip, like:

```
(--) SVGA: cldg5434: Specifying a Clocks line makes no sense for this driver
```

This example is taken from a `XF86_SVGA` server running a Cirrus Logic PCI card.

For boards which use programmable clock chips, you use a `ClockChip` line instead of a `Clocks` line in the `XF86Config` file. `ClockChip` is the name of the clock chip as used by the video card; the manual pages for each server describe them. For example,

in the file `README.S3`, we see that several S3-864 video cards use an "ICD2061A" clock chip, and that we should use the line

```
ClockChip "icd2061a"
```

instead of `Clocks` in the `XF86Config` file. As with `Clocks`, this line goes in the `Devices` section, after `Chipset`.

Similarly, some video cards require that you specify the RAMDAC chip type in the `XF86Config` file. This is done with a `Ramdac` line. The `XF86_Accel` man page describes this option. Often the X server will correctly probe for the RAMDAC.

Some video card types require that you specify several options in the `Devices` section of `XF86Config`. These options are described in the manual page for your server, as well as in the various files like `README.cirrus` and `README.S3`. These options are enabled using an `Option` line. For example, the #9 GXE 64 card requires two options:

```
Option "number_nine"
Option "dac_8_bit"
```

An X server may work without the `Option` lines, but they are necessary to get the best performance out of the card. There are too many options to list here. They are different for each card. If you must use one, the X server manual pages and various files in `/usr/X11R6/lib/X11/doc` will tell you what they are.

When you finish, you should have a `Devices` section that looks something like:

```
Section "Device"
        # Device section for the #9 GXE 64 only !
        Identifier "#9 GXE 64"
        Chipset "mmio_928"
        ClockChip "icd2061a"
        Option "number_nine"
        Option "dac_8_bit"
EndSection
```

There are other options which you can include in the `Devices` entry. The X server manual pages provide the gritty details.

5.7 Running XFree86.

With your `XF86Config` file configured, you can fire up the X server and give it a spin. Again, be sure that the `/usr/X11R6/bin` directory is on your path.

The command to start XFree86 is

```
startx
```

This is a front end to `xinit`. It starts the X server and executes the commands in the file `.xinitrc` in your home directory. `.xinitrc` is a shell script that contains the command lines of the X clients to run when the X server starts. If this file does not exist, the system default `/usr/X11R6/lib/X11/xinit/xinitrc` is used.

A simple `.xinitrc` file looks like this:

```
#!/bin/sh

xterm -fn 7x13bold -geometry 80x32+10+50 &
xterm -fn 9x15bold -geometry 80x34+30-10 &
oclock -geometry 70x70-7+7 &
xsetroot -solid midnightblue &

exec twm
```

This script starts two `xterm` clients and an `oclock`, and sets the root window (background) color to `midnightblue`. It starts `twm`, the window manager. `twm` is executed with the shell's `exec` statement. This causes the `xinit` process to be replaced by `twm`. After the `twm` process exits, the X server shuts down. You can cause `twm` to exit by using the root menu. Depress mouse button 1 on the desktop background. This displays a pop-up menu that allows you to `Exit Twm`.

Be sure that the last command in `.xinitrc` is started with `exec`, and that it is not placed into the background (no ampersand at the end of the line). Otherwise the X server will shut down immediately after it starts the clients in the `.xinitrc` file.

Alternately, you can exit X by pressing `Ctrl`-`Alt`-`Backspace` in combination. This kills the X server directly, exiting the window system.

The above is a only a simple desktop configuration. Again, we suggest that you read a book like *The X Window System: A User's Guide* (see Appendix A). The possible variations of X usage and configuration are too many to describe here. The `xterm`, `oclock`, and `twm` manual pages will provide you clues on how to begin.

5.8 When you run into trouble.

Often, something will not be quite right when you first start the X server. This is nearly always caused by something in your `XF86Config` file. Usually, the monitor timing values

or the video card dot clocks are set incorrectly. If the display seems to roll, or the edges are fuzzy, this indicates that the monitor timing values or dot clocks are wrong. Also, be sure that you correctly specified the video card chip set and options in the `Device` section of `XF86Config`. Be absolutely sure that you are using the correct X server and that `/usr/X11R6/bin/X` is a symbolic link to it.

If all else fails, try to start X "bare"; that is, with a command like:

```
X > /tmp/x.out 2>&1
```

You can then kill the X server (using Ctrl-Alt-Backspace) and examine the contents of `/tmp/x.out`. The X server reports any warnings or errors—for example, if your video card doesn't have a dot clock corresponding to a mode supported by your monitor.

The file `VideoModes.doc`, which is included in the XFree86 distribution, contains many hints for adjusting the values in your `XF86Config` file.

Remember that you can use Ctrl-Alt-Numeric + and Ctrl-Alt-Numeric - to switch between the video modes listed on the `Modes` line of the `Screen` section of `XF86Config`. If the highest resolution mode doesn't look right, try switching to a lower resolution. This lets you know, at least, that those parts of your X configuration are working correctly.

Also, adjust the vertical and horizontal size/hold knobs on your monitor. In many cases, it is necessary to adjust these when starting up X. For example, if the display seems to be shifted slightly to one side, you can usually correct this using the monitor controls.

Again, the USENET newsgroup `comp.windows.x.i386unix` is devoted to discussions about XFree86. It might be a good idea to read the newsgroups for postings related to video configuration. You might run across someone with the same problem.

There are also sample `XF86Config` files which have been contributed by users. Some of these are available on the `sunsite.unc.edu` archive in the `/pub/Linux/X11` directory, and elsewhere. You might find a configuration file that somebody has already written for your hardware.

Chapter 6

Networking

In this chapter we discuss Networking—how to configure a connection, using TCP/IP, SLIP, PPP or UUCP, and electronic mail and news.

6.1 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.

The current implementation of TCP/IP and related protocols for Linux is called "NET-3," and before that, "NET-2." This has no relationship to the so-called NET-2 release of BSD UNIX; instead, "NET-3" in this context means the second implementation of TCP/IP for Linux.

Linux NET-3 also supports SLIP—Serial Line Internet Protocol and PPP—Point-to-Point Protocol. SLIP and PPP allow you to have dialup Internet access using a modem. If your business or university provides SLIP or PPP access, you can dial in to the SLIP or PPP 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 or PPP server.

For complete information on setting up TCP/IP under Linux, we encourage you to read the Linux NET-3 HOWTO, available via anonymous FTP from `sunsite.unc.edu`. The NET-3 HOWTO is a complete guide to configuring TCP/IP, including Ethernet and SLIP or PPP connections, under Linux. The Linux Ethernet HOWTO is a related document that describes configuration of various Ethernet card drivers for Linux. The *Linux Network Administrator's Guide*, from the Linux Documentation Project, is also available. See Appendix A for more information on these documents.

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.

TCP/IP 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 an Ethernet card. Common cards such as the 3com 3c503, HP PCLAN (27245 and 27xxx series), Western Digital WD80x3, and Novell NE2000/NE1000 are supported, as well as many more. See the Linux Ethernet and Hardware HOWTOs for details.

There are a few common situations that you should watch out concerning supported cards: 1) Several cards are support but offer shoddy performance or have other restrictions. Examples are the 3Com 3C501 which works but gives absolutely horrible performance and the Racal-Interlan NI6510 using the am7990 lance chip which doesn't work with more than 16 megabytes of RAM. In the same vein, many cards are NE1000/NE2000 compatible clones and can have various problems. See the Linux Ethernet HOWTO for a more complete discussion of Linux Ethernet hardware compatibility.

Linux also supports SLIP and PPP, 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 or PPP server—most servers require a 14.4bps V.32bis modem at a minimum . Performance is greatly improved with a 33.6bps or higher modem.

6.1.1 Configuring TCP/IP on your system.

In this section we're going to discuss how to configure an Ethernet TCP/IP connection on your system. Note that this method should work for many systems, but certainly not all.

This discussion should be enough to get you on the right path to configuring the network parameters of your machine, but there are numerous caveats and fine details not mentioned here. We direct you to the *Linux Network Administrators' Guide* and the NET-3-HOWTO for more information.[1]

First, we assume that you have a Linux system that has the TCP/IP software installed. This includes basic clients such as `telnet` and `ftp`, system administration commands such as `ifconfig` and `route` (usually found in `/etc`), and networking configuration files (such as `/etc/hosts`). The other Linux-related networking documents described above explain how to go about installing the Linux networking software if you do not have it already.

We also assume that your kernel has been configured and compiled with TCP/IP support enabled. See Section 4.9 for information on compiling your kernel. To enable networking, you must answer "yes" to the appropriate questions during the `make config` step, and rebuild the kernel.

Once this has been done, you must modify a number of configuration files used by NET-3. For the most part this is a simple procedure. Unfortunately, however, there is wide disagreement between Linux distributions as to where the various TCP/IP configuration files and support programs should go. Much of the time, they can be found in `/etc`, but in other cases may be found in `/usr/etc`, `/usr/etc/inet`, or other bizarre locations. In the worst case you'll have to use the `find` command to locate the files on your system. Also note that not all distributions keep the NET-3 configuration files and software in the same location—they may be spread across several directories.

The following information applies primarily to Ethernet connections. If you're planning to use SLIP or PPP, read this section to understand the concepts, and follow the more specific instructions in the following sections.

Your network configuration. Before you can configure TCP/IP, you need to determine the following information about your network setup. In most cases, your local network administrator can provide you with this information.

- IP address. This is the unique machine address in dotted-decimal format. An example is 128.253.153.54. Your network admins will provide you with this number.

  If you're only configuring loopback mode (i.e. no SLIP, no Ethernet card, just TCP/IP connections to your own machine) then your IP address is 127.0.0.1.

- Your network mask ("netmask"). This is a dotted quad, similar to the IP address,

[1] Some of this information is adapted from the NET-3-HOWTO by Terry Dawson and Matt Welsh.

which determines which portion of the IP address specifies the subnetwork number, and which portion specifies the host on that subnet. (If you're shaky on these TCP/IP networking terms, we suggest reading some introductory material on network administration.) The network mask is a pattern of bits, which when overlayed onto an address on your network, will tell you which subnet that address lives on. This is very important for routing, and if you find, for example, that you can happily talk to people outside your network, but not to some people within your network, there is a good chance that you have an incorrect mask specified.

Your network administrators will have chosen the netmask when the network was designed, and therefore they should be able to supply you with the correct mask to use. Most networks are class C subnetworks which use 255.255.255.0 as their netmask. Class B networks use 255.255.0.0. The NET-3 code will automatically select a mask that assumes no subnetting as a default if you do not specify one.

This applies as well to the loopback port. Since the loopback port's address is always 127.0.0.1, the netmask for this port is always 255.0.0.0. You can either specify this explicitly or rely on the default mask.

- Your network address. This is your IP address masked bitwise-ANDed the netmask. For example, if your netmask is 255.255.255.0, and your IP address is 128.253.154.32, your network address is 128.253.154.0. With a netmask of 255.255.0.0, this would be 128.253.0.0.

  If you're only using loopback, you don't have a network address.

- Your broadcast address. The broadcast address is used to broadcast packets to every machine on your subnet. Therefore, if the host number of machines on your subnet is given by the last byte of the IP address (netmask 255.255.255.0), your broadcast address will be your network address ORed with 0.0.0.255.

  For example, if your IP address is 128.253.154.32, and your netmask is 255.255.255.0, your broadcast address is 128.253.154.255.

  Note that for historical reasons, some networks are setup to use the network address as the broadcast address, if you have any doubt, check with your network administrators. (In many cases, it will suffice to duplicate the network configuration of other machines on your subnet, substituting your own IP address, of course.)

  If you're only using loopback, you don't have a broadcast address.

- Your gateway address. This is the address of the machine which is your "gateway" to the outside world (i.e. machines not on your subnet). In many cases the gateway machine has an IP address identical to yours but with a ".1" as its host address; e.g., if your IP address is 128.253.154.32, your gateway might be 128.253.154.1. Your network admins will provide you with the IP address of your gateway.

  In fact, you may have multiple gateways. A *gateway* is simply a machine that lives on two different networks (has IP addresses on different subnets), and routes packets between them. Many networks have a single gateway to "the outside world" (the network directly adjacent to your own), but in some cases you will have multiple gateways—one for each adjacent network.

  If you're only using loopback, you don't have a gateway address. The same is true if your network is isolated from all others.

- Your name server address. Most machines on the net have a name server which translates host names into IP addresses for them. Your network admins will tell you the address of your name server. You can also run a server on your own machine by running `named`, in which case the name server address is 127.0.0.1. Unless you absolutely *must* run your own name server, we suggest using the one provided to you on the network (if any). Configuration of `named` is another issue altogether; our priority at this point is to get you talking to the network. You can deal with name resolution issues later.

  If you're only using loopback, you don't have a name server address.

SLIP/PPP users: You may or may not require any of the above information, except for a name server address. When using SLIP, your IP address is usually determined in one of two ways: Either (a) you have a "static" IP address, which is the same every time you connect to the network, or (b) you have a "dynamic" IP address, which is allocated from a pool available addresses when you connect to the server. In the following section on SLIP configuration this is covered in more detail.

NET-3 supports full routing, multiple routes, subnetworking (at this stage on byte boundaries only), the whole nine yards. The above describes most basic TCP/IP configurations. Yours may be quite different: when in doubt, consult your local network gurus and check out the man pages for `route` and `ifconfig`. Configuring TCP/IP networks is very much beyond the scope of this book; the above should be enough to get most people started.

The networking `rc` files. `rc` files are systemwide configuration scripts executed at boot time by `init`, which start up all of the basic system daemons (such as `sendmail`, `cron`, etc.) and configure things such as the network parameters, system host name, and so on. `rc` files are usually found in the directory `/etc/rc.d` but on other systems may be in `/etc`. In general Slackware distributions use the files `rc.inet1`, etc. in `/etc/rc.d` whereas the RedHat distributions use a series of directories

Here, we're going to describe the `rc` files used to configure TCP/IP. There are two of them: `rc.inet1` and `rc.inet2`. `rc.inet1` is used to configure the basic network parameters (such as IP addresses and routing information) and `rc.inet2` fires up the TCP/IP daemons (`telnetd`, `ftpd`, and so forth).

Many systems combine these two files into one, usually called `rc.inet` or `rc.net`. The names given to your `rc` files doesn't matter, as long as they perform the correct functions and are executed at boot time by `init`. To ensure this, you may need to edit `/etc/inittab` and uncomment lines to execute the appropriate `rc` file(s). In the worst case you will have to create the `rc.inet1` and `rc.inet2` files from scratch and add entries for them to `/etc/inittab`.

As we said, `rc.inet1` configures the basic network interface. This includes your IP and network address, and the routing table information for your network. The routing tables are used to route outgoing (and incoming) network datagrams to other machines. On most simple configurations, you have three routes: One for sending packets to your own machine, another for sending packets to other machines on your network, and another for sending packets to machines outside of your network (through the gateway machine). Two programs are used to configure these parameters: `ifconfig` and `route`. Both of these are usually found in `/etc`.

`ifconfig` is used for configuring the network device interface with the parameters that it requires to function, such as the IP address, network mask, broadcast address and the like. `Route` is used to create and modify entries in the routing table.

For most configurations, an `rc.inet1` file that looks like the following should work. You will, of course, have to edit this for your own system. Do *not* use the sample IP and network addresses listed here for your own system; they correspond to an actual machine on the Internet.

```
#!/bin/sh
# This is /etc/rc.d/rc.inet1 -- Configure the TCP/IP interfaces

# First, configure the loopback device
```

```
HOSTNAME=‘host name‘

/etc/ifconfig lo 127.0.0.1      # uses default netmask 255.0.0.0
/etc/route add 127.0.0.1        # a route to point to the loopback device

# Next, configure the ethernet device. If you’re only using loopback or
# SLIP, comment out the rest of these lines.

# Edit for your setup.
IPADDR="128.253.154.32"         # REPLACE with YOUR IP address
NETMASK="255.255.255.0"         # REPLACE with YOUR netmask
NETWORK="128.253.154.0"         # REPLACE with YOUR network address
BROADCAST="128.253.154.255"     # REPLACE with YOUR broadcast address, if you
                                # have one. If not, leave blank and edit below.
GATEWAY="128.253.154.1"         # REPLACE with YOUR gateway address!

/etc/ifconfig eth0 ${IPADDR} netmask ${NETMASK} broadcast ${BROADCAST}

# If you don’t have a broadcast address, change the above line to just:
# /etc/ifconfig eth0 ${IPADDR} netmask ${NETMASK}

/etc/route add ${NETWORK}

# The following is only necessary if you have a gateway; that is, your
# network is connected to the outside world.
/etc/route add default gw ${GATEWAY} metric 1

# End of Ethernet Configuration
```

Again, you may have to tweak this file somewhat to get it to work. The above should be sufficient for the majority of simple network configurations, but certainly not all.

`rc.inet2` starts up various servers used by the TCP/IP suite. The most important of these is `inetd`. `Inetd` sits in the background and listens to various network ports. When a machine tries to make a connection to a certain port (for example, the incoming `telnet` port), `inetd` forks off a copy of the appropriate daemon for that port (in the case of the `telnet` port, `inetd` starts `in.telnetd`). This is simpler than running many separate, standalone daemons (e.g., individual copies of `telnetd`, `ftpd`, and so forth)—`inetd` starts up the daemons only when they are needed.

`Syslogd` is the system logging daemon—it accumulates log messages from vari-

ous applications and stores them into log files based on the configuration information in `/etc/syslogd.conf`. `routed` is a server used to maintain dynamic routing information. When your system attempts to send packets to another network, it may require additional routing table entries in order to do so. `routed` takes care of manipulating the routing table without the need for user intervention.

Our example `rc.inet2`, below, only starts up the bare minimum of servers. There are many other servers as well—many of which have to do with NFS configuration. When attempting to setup TCP/IP on your system, it's usually best to start with a minimal configuration and add more complex pieces (such as NFS) when you have things working.

Note that in the below file, we assume that all of the network daemons are held in `/etc`. As usual, edit this for your own configuration.

```
#! /bin/sh
# Sample /etc/rc.d/rc.inet2

# Start syslogd
if [ -f /etc/syslogd ]
then
      /etc/syslogd
fi

# Start inetd
if [ -f /etc/inetd ]
then
      /etc/inetd
fi

# Start routed
if [ -f /etc/routed ]
then
      /etc/routed -q
fi

# Done!
```

Among the various additional servers that you may want to start in `rc.inet2` is `named`. `Named` is a name server—it is responsible for translating (local) IP addresses to names, and vice versa. If you don't have a name server elsewhere on the network, or want to provide local machine names to other machines in your domain, it may be necessary to

run `named`. (For most configurations it is not necessary, however.) `Named` configuration is somewhat complex and requires planning; we refer interested readers to a good book on TCP/IP network administration.

The `/etc/hosts` file. `/etc/hosts` contains a list of IP addresses and the host names that they correspond to. In general, `/etc/hosts` only contains entries for your local machine, and perhaps other "important" machines (such as your name server or gateway). Your local name server will provide address-to-name mappings for other machines on the network, transparently.

For example, if your machine is `loomer.vpizza.com` with the IP address 128.253.154.32, your `/etc/hosts` would look like:

```
127.0.0.1               localhost
128.253.154.32          loomer.vpizza.com loomer
```

If you're only using loopback, the only line in `/etc/hosts` should be for 127.0.0.1, with both `localhost` and your host name after it.

The `/etc/networks` file. The `/etc/networks` file lists the names and addresses of your own, and other, networks. It is used by the `route` command, and allows you to specify a network by name, should you so desire.

Every network you wish to add a route to using the `route` command (generally called from `rc.inet1`—see above) *must* have an entry in `/etc/networks`.

As an example,

```
default 0.0.0.0 # default route    - mandatory
loopnet 127.0.0.0 # loopback network - mandatory
mynet 128.253.154.0 # Modify for your own network address
```

The `/etc/host.conf` file. This file is used to specify how your system will resolve host names. It should contain the two lines:

```
order hosts,bind
multi on
```

These lines tell the resolve libraries to first check the `/etc/hosts` file for any names to lookup, and then to ask the name server (if one is present). The `multi` entry allows you to have multiple IP addresses for a given machine name in `/etc/hosts`.

The `/etc/resolv.conf` file. This file configures the name resolver, specifying the address of your name server (if any) and your domain name. Your domain name is your fully-qualified host name (if you're a registered machine on the Internet, for example), with the host name chopped off. That is, if your full host name is `loomer.vpizza.com`, your domain name is just `vpizza.com`.

For example, if your machine is `goober.norelco.com`, and has a name server at the address 128.253.154.5, your `/etc/resolv.conf` would look like:

```
domain      norelco.com
nameserver  127.253.154.5
```

You can specify more than one name server—each must have a `nameserver` line of its own in `resolv.conf`.

Setting your host name. You should set your system host name with the `hostname` command. This is usually called from `/etc/rc` or `/etc/rc.local`; simply search your system `rc` files to determine where it is invoked. For example, if your (full) host name is `loomer.vpizza.com`, edit the appropriate `rc` file to execute the command:

```
/bin/hostname loomer.vpizza.com
```

Note that the `hostname` executable may not be found in `/bin` on your system.

Trying it out. Once you have all of these files set up, you should be able to reboot your new kernel and attempt to use the network. There are many places where things can go wrong, so it's a good idea to test individual aspects of the network configuration (e.g., it's probably not a good idea to test your network configuration by firing up Mosaic over a network-based X connection).

You can use the `netstat` command to display your routing tables; this is usually the source of the most trouble. The `netstat` man page describes the exact syntax of this command in detail. In order to test network connectivity, we suggest using a client such as `telnet` to connect to machines both on your local subnetwork and external networks. This will help to narrow down the source of the problem. (For example, if you're unable to connect to local machines, but can connect to machines on other networks, more than likely there is a problem with your netmask and routing table configuration). You can also invoke the `route` command directly (as `root`) to play with the entries in your routing table.

You should also test network connectivity by specifying IP addresses directly, instead of host names. For example, if you have problems with the command

```
$ telnet shoop.vpizza.com
```

the cause may be incorrect name server configuration. Try using the actual IP address of the machine in question; if that works, then you know your basic network setup is (more than likely) correct, and the problem lies in your specification of the name server address.

Debugging network configurations can be a difficult task, and we can't begin to cover it here. If you are unable to get help from a local guru we strongly suggest reading the *Linux Network Administrators' Guide* from the LDP.

6.1.2 SLIP configuration.

SLIP (Serial Line Internet Protocol) allows you to use TCP/IP over a serial line, be that a phone line, with a dialup modem, or a leased asynchronous line of some sort. Of course, to use SLIP you'll need access to a dial-in SLIP server in your area. Many universities and businesses provide SLIP access for a modest fee.

There are two major SLIP-related programs available—`dip` and `slattach`. Both of these programs are used to initiate a SLIP connection over a serial device. It is *necessary* to use one of these programs in order to enable SLIP—it will not suffice to dial up the SLIP server (with a communications program such as `kermit`) and issue `ifconfig` and `route` commands. This is because `dip` and `slattach` issue a special *ioctl()* system call to seize control of the serial device to be used as a SLIP interface.

`dip` can be used to dial up a SLIP server, do some handshaking to login to the server (exchanging your username and password, for example) and then initiate the SLIP connection over the open serial line. `slattach`, on the other hand, does very little other than grab the serial device for use by SLIP. It is useful if you have a permanent line to your SLIP server and no modem dialup or handshaking is necessary to initiate the connection. Most dialup SLIP users should use `dip`, on the other hand.

`dip` can also be used to configure your Linux system as a SLIP server, where other machines can dial into your own and connect to the network through a secondary Ethernet connection on your machine. See the documentation and man pages for `dip` for more information on this procedure.

SLIP is quite unlike Ethernet, in that there are only two machines on the "network"—the SLIP host (that's you) and the SLIP server. For this reason, SLIP is often referred to as a "point-to-point" connection. A generalization of this idea, known as PPP (Point to Point Protocol) has also been implemented for Linux.

When you initiate a connection to a SLIP server, the SLIP server will give you an IP address based on (usually) one of two methods. Some SLIP servers allocate "static" IP

addresses—in which case your IP address will be the same every time you connect to the server. However, many SLIP servers allocate IP addresses dynamically—in which case you receive a different IP address each time you connect. In general, the SLIP server will print the values of your IP and gateway addresses when you connect. `dip` is capable of reading these values from the output of the SLIP server login session and using them to configure the SLIP device.

Essentially, configuring a SLIP connection is just like configuring for loopback or ethernet. The main differences are discussed below. Read the previous section on configuring the basic TCP/IP files, and apply the changes described below.

Static IP address SLIP connections using `dip`. If you are using a static-allocation SLIP server, you may want to include entries for your IP address and host name in `/etc/hosts`. Also, configure these files listed in the above section: `rc.inet2`, `host.conf`, and `resolv.conf`.

Also, configure `rc.inet1`, as described above. However, you only want to execute `ifconfig` and `route` commands for the loopback device. If you use `dip` to connect to the SLIP server, it will execute the appropriate `ifconfig` and `route` commands for the SLIP device for you. (If you're using `slattach`, on the other hand, you *will* need to include `ifconfig`/`route` commands in `rc.inet1` for the SLIP device—see below.)

`dip` *should* configure your routing tables appropriately for the SLIP connection when you connect. In some cases, however, `dip`'s behavior may not be correct for your configuration, and you'll have to run `ifconfig` or `route` commands by hand after connecting to the server with `dip` (this is most easily done from within a shell script that runs `dip` and immediately executes the appropriate configuration commands). Your gateway is, in most cases, the address of the SLIP server. You may know this address before hand, or the gateway address will be printed by the SLIP server when you connect. Your `dip` chat script (described below) can obtain this information from the SLIP server.

`ifconfig` may require use of the `pointopoint` argument, if `dip` doesn't configure the interface correctly. For example, if your SLIP server address is 128.253.154.2, and your IP address is 128.253.154.32, you may need to run the command

```
ifconfig sl0 128.253.154.32 pointopoint 128.253.154.2
```

as `root`, after connecting with `dip`. The man pages for `ifconfig` will come in handy.

Note that SLIP device names used with the `ifconfig` and `route` commands are `sl0`, `sl1` and so on (as opposed to `eth0`, `eth1`, etc. for Ethernet devices).

In Section 6.1.2, below, we explain how to configure `dip` to connect to the SLIP server.

Static IP address SLIP connections using `slattach`. If you have a leased line or cable running directly to your SLIP server, then there is no need to use `dip` to initiate a connection. `slattach` can be used to configure the SLIP device instead.

In this case, your `/etc/rc.inet1` file should look something like the following:

```
#!/bin/sh
IPADDR="128.253.154.32"         # Replace with your IP address
REMADDR="128.253.154.2"
# Replace with your SLIP server address

# Modify the following for the appropriate serial device for the SLIP
# connection:
slattach -p cslip -s 19200 /dev/ttyS0
/etc/ifconfig sl0 $IPADDR pointopoint $REMADDR up
/etc/route add default gw $REMADDR
```

`slattach` allocates the first unallocated SLIP device (`sl0`, `sl1`, etc.) to the serial line specified.

Note that the first parameter to `slattach` is the SLIP protocol to use. At present the only valid values are `slip` and `cslip`. `Slip` is regular SLIP, as you would expect, and `cslip` is SLIP with datagram header compression. In most cases you should use `cslip`; however, if you seem to be having problems with this, try `slip`.

If you have more than one SLIP interface then you will have routing considerations to make. You will have to decide what routes to add, and those decisions can only be made on the basis of the actual layout of your network connections. A book on TCP/IP network configuration, as well as the man pages to `route`, will be of use.

Dynamic IP address SLIP connections using `dip`. If your SLIP server allocates an IP address dynamically, then you certainly don't know your address in advance—therefore, you can't include an entry for it in `/etc/hosts`. (You should, however, include an entry for your host with the loopback address, 127.0.0.1.)

Many SLIP servers print your IP address (as well as the server's address) when you connect. For example, one type of SLIP server prints a string such as,

```
Your IP address is 128.253.154.44.
Server address is 128.253.154.2.
```

`dip` can capture these numbers from the output of the server and use them to configure the SLIP device.

See page 261, above, for information on configuring your various TCP/IP files for use with SLIP. Below, we explain how to configure `dip` to connect to the SLIP server.

Using `dip`.

`dip` can simplify the process of connecting to a SLIP server, logging in, and configuring the SLIP device. Unless you have a leased line running to your SLIP server, `dip` is the way to go.

To use `dip`, you'll need to write a "chat script" which contains a list of commands used to communicate with the SLIP server at login time. These commands can automatically send your user name/password to the server, as well as get information on your IP address from the server.

Here is an example `dip` chat script, for use with a dynamic IP address server. For static servers, you will need to set the variables `$local` and `$remote` to the values of your local IP address and server IP address, respectively, at the top of the script. See the `dip` man page for details.

```
main:
  # Set Maximum Transfer Unit. This is the maximum size of packets
  # transmitted on the SLIP device. Many SLIP servers use either 1500 or
  # 1006; check with your network admins when in doubt.
  get $mtu 1500

  # Make the SLIP route the default route on your system.
  default

  # Set the desired serial port and speed.
  port cua03
  speed 38400

  # Reset the modem and terminal line. If this causes trouble for you,
  # comment it out.
  reset

  # Prepare for dialing. Replace the following with your
  # modem initialization string.
  send ATT&C1&D2\\N3&Q5%M3%C1N1W1L1S48=7\r
  wait OK 2
  if $errlvl != 0 goto error
```

```
  # Dial the SLIP server
  dial 2546000
  if $errlvl != 0 goto error
  wait CONNECT 60
  if $errlvl != 0 goto error

  # We are connected.  Login to the system.
login:
  sleep 3
  send \r\n\r\n
  # Wait for the login prompt
  wait login: 10
  if $errlvl != 0 goto error

  # Send your username
  send USERNAME\n

  # Wait for password prompt
  wait ord: 5
  if $errlvl != 0 goto error

  # Send password.
  send PASSWORD\n

  # Wait for SLIP server ready prompt
  wait annex: 30
  if $errlvl != 0 goto error

  # Send commands to SLIP server to initiate connection.
  send slip\n
  wait Annex 30

  # Get the remote IP address from the SLIP server. The 'get...remote'
  # command reads text in the form xxx.xxx.xxx.xxx, and assigns it
  # to the variable given as the second argument (here, $remote).
  get $remote remote
  if $errlvl != 0 goto error
  wait Your 30
```

```
    # Get local IP address from SLIP server, assign to variable $local.
    get $local remote
    if $errlvl != 0 goto error

    # Fire up the SLIP connection
  done:
    print CONNECTED to $remote at $rmtip
    print GATEWAY address $rmtip
    print LOCAL address $local
    mode SLIP
    goto exit
  error:
    print SLIP to $remote failed.

  exit:
```

`dip` automatically executes `ifconfig` and `route` commands based on the values of the variables `$local` and `$remote`. Here, those variables are assigned using the `get...remote` command, which obtains text from the SLIP server and assigns it to the named variable.

If the `ifconfig` and `route` commands that `dip` runs for you don't work, you can either run the correct commands in a shell script after executing `dip`, or modify the source for `dip` itself. Running `dip` with the `-v` option will print debugging information while the connection is being set up, which should help you to determine where things might be going awry.

Now, in order to run `dip` and open the SLIP connection, you can use a command such as:

```
/etc/dip/dip -v /etc/dip/mychat 2>&1
```

Where the various `dip` files, and the chat script (`mychat.dip`), are stored in `/etc/dip`.

The above discussion should be enough to get you well on your way to talking to the network, either via Ethernet or SLIP. Again, we strongly suggest looking into a book on TCP/IP network configuration, especially if your network has any special routing considerations, other than those mentioned here.

6.2 Dial-up networking and PPP.

Linux supports a full implementation of the PPP (Point-to-Point) networking protocols. PPP is a mechanism for creating and running IP (the Internet Protocol) and other network protocols over a serial connection (using a null-modem cable), over a telnet established link or a link made using modems and telephone lines (and of course using digital lines such as ISDN). This section will only cover configuring PPP as a client connecting via an analog modem to a remote machine that provides PPP dialup service.

For complete information on setting up PPP under Linux, we encourage you to read the Linux PPP HOWTO, available via anonymous FTP from `sunsite.unc.edu`. The PPP HOWTO is a complete guide to configuring PPP, including modem, ISDN and null-modem cables, under Linux. Much of the information in this section was gleaned from this document. The *Linux Network Administrator's Guide*, from the Linux Documentation Project, is also available. See Appendix A for more information on these documents.

6.2.1 What you need to get started.

We assume that your kernel has been configured and compiled with TCP/IP support enabled. See Section 4.9 for information on compiling your kernel. To enable networking, you must answer "yes" to the appropriate questions during the `make config` step, and rebuild the kernel. We also assume that ppp has been compiled and installed on your system as well. We assume that you are using a Linux 1.2.x kernel with the PPP 2.1.2 software or Linux 1.3.X/2.0.x and PPP 2.2.0. At the time of writing, the latest official version of PPP available for Linux is ppp-2.2f. Please see the kerneld mini-HOWTO if you plan to use modules to load ppp into your kernel. *It is highly recommended that you use a version of the Linux kernel and the appropriate PPP version that are known to be stable together.*

You should also read

- the documentation that comes with the PPP package;
- the pppd and chat man pages; (use *man chat* and *man pppd* to explore these)
- the Linux Network Administration Guide (NAG);
- the Net-2/3 HOWTO;
- Linux kernel documentation installed in /usr/src/linux/Documentation when you install the Linux source code;

- The modem setup information page—see Modem Setup Information (http://www.in.net/info/modems/index.html)
- The excellent Unix/Linux books published by O'Reilly and Associates. See (O'Reilly and Associates On-Line catalog (`http://www.ora.com/`). If you are new to Unix/Linux, run (don't walk) to your nearest computer book shop and invest in a number of these immediately!
- The PPP-FAQ maintained by Al Longyear, available from (`ftp://sunsite.unc.edu/pub/Linux/docs/faqs`; see Appendix B). This contains a great deal of useful information in question/answer format that is very useful when working out why PPP is not working (properly).

6.2.2 An overview of the steps involved.

There are several steps to setting up your system to use PPP. We recommend that you read through all of these steps thoroughly before attempting to actually bring up a PPP connection. Each of these steps will be discussed in detail later.

1. Make sure that TCP/IP support is compiled into your kernel.
2. Make sure that PPP support is compiled into your kernel either statically or as a loadable module.
3. Make sure that PPP software is compiled and installed on your systems.
4. Make sure that you have a modem configured and installed/attached to your computer and that you know which serial port the modem is assigned to.
5. Make sure you have the following information from your PPP dialup server provider (usually an ISP)
    - The phone number you will dial to connect to the remote PPP dialup service provider.
    - Whether or not you are using dynamic or static IP assignment. If the latter, you will need to know that static IP number.
    - The IP addresss of the DNS (Domain Name Service) server that you will be using to resolve host names when connected

Make sure that the kernel has TCP/IP support. Linux PPP operations come in two parts: 1) the PPP daemon and kernel support for PPP. Most distributions seem to provide PPP kernel support in their default installation kernels, but others do not. You should make sure that TCP/IP is compiled into your kernel. You can do this by issuing the following command:

```
grep -i ‘‘TCP/IP’’ /var/adm/messages
```

If you get a line similar to

```
Jun  8 09:52:08 gemini kernel: Swansea University Computer Society TCP/IP for N
```

then you have TCP/IP support compiled in. You can also look for the above information on the console while Linux is booting. On many fast machines, this scrolls by too quickly. You can use Shift-PageUp to scroll the screen up and see this.

Make sure that the kernel has PPP support. If at boot your kernel reports messages like

```
PPP Dynamic channel allocation code copyright 1995
Caldera, Inc.
PPP line discipline registered.
```

then your kernel has PPP support. You can also issue the command

```
# grep -i ‘‘PPP’’ /var/adm/messages
```

If you get a line similar to

```
Jun  8 09:52:08 gemini kernel: PPP: version 0.2.7 (4 channels) NEW_TTY_DRIVERS
```

that means PPP support is present.

Make sure that you have a modem configured. You should make sure that your modem is correctly set up and that you know which serial port it is connected to.

- DOS com1: = Linux /dev/cua0 (and /dev/ttyS0)
- DOS com2: = Linux /dev/cua1 (and /dev/ttyS1),
- et cetera

Historically, Linux used cua*x* devices for dial out and ttyS*x* devices for dial in. The kernel code that required this was changed in kernel version 2.0.*x* and you should now use ttyS*x* for both dial in and dial out. The cua*x* device names may well disappear in future kernel versions.

If you are using a high speed (external) modem (14,400 Baud or above), your serial port needs to be capable of handling the throughput that such a modem is capable of producing, particularly when the modems are compressing the data.

This requires your serial port to use a modern UART (Universal Asynchronous Receiver Transmitter) such as a 16550A. If you are using an old machine (or old serial card), it is quite possible that your serial port has only an 8250 UART, which will cause you considerable problems when used with a high speed modem.

Use the command

```
# setserial -a /dev/ttySx
```

to get Linux to report to you the type of UART you have. If you do not have a 16550A type UART, invest in a new serial card (available for under $50).

You will need to configure your modem correctly for PPP—to do this READ YOUR MODEM MANUAL! Most modems come with a factory default setting that selects the options required for PPP. Recommended configuration specifies (in standard Hayes commands):

- Hardware flow control (RTS/CTS) (&K3 on many modems)
- E1 Command/usr/src/linux-2.0.27/include/linux/serial.h Echo ON (required for chat to operate)
- Q0 Report result codes (required for chat to operate)
- S0=0 Auto Answer OFF (unless you want your modem to answer the phone)
- &C1 Carrier Detect ON only after connect
- &S0 Data Set Ready (DSR) always ON
- (depends) Data Terminal Ready

There is a site offering sample configurations for a growing variety of modem makes and models at Modem setup information (http://www.in.net/info/modems/index.html) which may assist you in this.

Use your communications software (e.g. minicom or seyon) to find out about your modem configuration and set it to what is required for PPP. Many modems report their current settings in response to AT&V, but you should consult your modem manual.

If you completely mess up the settings, you can return to sanity (usually) by issuing an AT&F—return to factory settings. (For most modem modems I have encountered, the factory settings include all you need for PPP—but you should check).

Once you have worked out the modem setup string required write it down. You now have a decision: you can store these settings in your modem non-volatile memory so they can be recalled by issuing the appropriate AT command. Alternatively you can pass the correct settings to your modem as part of the PPP dialing process.

If you only use your modem from Linux to call into your ISP or corporate server, the simplest set up will have you save your modem configuration in non-volatile RAM.

If on the other hand, your modem is used by other applications and operating systems, it is safest to pass this information to the modem as each call is made so that the modem is guaranteed to be in the correct state for the call. (This has the added advantage also of recording the modem setup string in case the modem looses the contents of its NV-RAM, which can indeed happen).

ISP information. Before you can establish your PPP connection with a remote server, you need to obtain the following information from the system administrator or technical support people of the ISP.

- The telephone number(s) to dial for the service If you are behind a PABX, you also need the PABX number that gives you an outside dial tone—this is frequently digit zero (0) or nine (9).

- Does the server use DYNAMIC or STATIC IP numbers? If the server uses STATIC IP numbers, then you may need to know what IP number to use for your end of the PPP connection. If your ISP is providing you with a subnet of valid IP numbers, you will need to know the IP numbers you can use and the network mask (netmask).

  Most Internet Service Providers use DYNAMIC IP numbers. As mentioned above, this has some implications in terms of the services you can use.

  However, even if you are using STATIC IP numbers, most PPP servers will never (for security reasons) allow the client to specify an IP number as this is a security risk. You do still need to know this information!

- What are the IP numbers of the ISPs Domain Name Servers? There should be at least two although only one is needed.

  There could be a problem here. The MS Windows 95 PPP setup allows the DNS address to be passed to the client as part of its connection process. So your ISP (or corporate help desk) may well tell you you don't need the IP address of the DNS server(s).

  For Linux, you DO need the address of at least one DNS. The linux implementation of PPP does not allow the setting of the DNS IP number dynamically at connection time—and quite possibly will never do so.

- Does the server require the use of PAP/CHAP? If this is the case you need to know the "id" and "secret" you are to use in connecting. (These are probably your user name and password at your ISP).

- Does the server automatically start PPP or do you need to issue any commands to start PPP on the server once you are logged in? If you must issue a command to start PPP, what is it?

- Is the server a Microsoft Windows NT system and, if so, is it using the MS PAP/CHAP system? Many corporate LANs seem to use MS Windows NT this way for increased security.

Every device that connects to the Internet must have its own, unique IP number. These are assigned centrally by a designated authority for each country. Therefore to use a PPP connection must have an IP assigned to you. Due to the increased number of machines on the Internet (partly do the large number of PPP users), a dynamic scheme has been developed for PPP that provides an IP on the fly to your machine when it first establishes the PPP connection. This means that you will have a different IP address every time you connect to the remote PPP dialup service. This is the most common method for most ISPs. The other method is to use a static IP. You cannot just choose an IP to us. It must be assigned by the centralized agency in charge of issuing IP numbers. This prevents two computers from having the same IP address and causing problems on the Internet. The remote PPP dialup service provider will be able to tell you if you are using a static or dynamic IP and also provide you with the actual IP number if you are using the static method.

It is important to note that if you are using dynamic IP assignment, it will be very very difficult to provide any permanent Internet services such as World Wide Web servers, gopher services, or Internet Relay Chat servers. You can still use such services that are on

other machines but cannot offer such services on your machine without going through an extreme amount of effort. Doing this beyond the scope of this document.

PAP and CHAP are different commonly-used authentication methods. Linux supports both of them.

Testing your modem and remote service. Now that you have sorted out the serial port and modem settings it is a good idea to make sure that these setting do indeed work by dialing you ISP and seeing if you can connect.

Using you terminal communications package (such as minicom or seyon), set up the modem initialization required for PPP and dial into the PPP server you want to connect to with a PPP session.

(Note: at this stage we are NOT trying to make a PPP connection—just establishing that we have the right phone number and also to find out exactly what the server sends to us in order to get logged in and start PPP).

During this process, either capture (log to a file) the entire login process or carefully (very carefully) write down exactly what prompts the server gives to let you know it is time to enter your user name and password (and any other commands needed to establish the PPP connection).

If your server uses PAP, you should not see a login prompt, but should instead see the (text representation) of the link control protocol (which looks like garbage) starting on your screen.

A few words of warning:

- some servers are quite intelligent: you can log in using text based user name/passwords OR using PAP. So if your ISP or corporate site uses PAP but you do not see the garbage start up immediately, this may not mean you have done something wrong.
- some servers require you to enter some text initially and then start a standard PAP sequence.
- Some PPP servers are passive—that is they simply sit there sending nothing until the client that is dialing in sends them a valid lcp packet. If the PPP server you are connecting to operates in passive mode, you will never see the garbage!
- Some servers do not start PPP until you press ENTER—so it is worth trying this if you correctly log in and do not see the garbage!

It is worth dialing in at least twice—some servers change their prompts (e.g. with the time!) every time you log in. The two critical prompts your Linux box needs to be able to identify every time you dial in are:

- the prompt that requests you to enter your user name;
- the prompt that requests you to enter your password;

If you have to issue a command to start PPP on the server, you will also need to find out the prompt the server gives you once you are logged in to tell you that you can now enter the command to start PPP.

If your server automatically starts PPP, once you have logged in, you will start to see garbage on your screen—this is the PPP server sending your machine information to start up and configure the PPP connection.

This should look something like this :

```
y}#.!}!}!} }8}!}}U}"}\&} } } } }}\& ...}'}"}(}"} .~~y}
```

On some systems PPP must be explicitly started on the server. This is usually because the server has been set up to allow PPP logins and shell logins using the same user name/password pair. If this is the case, issue this command once you have logged in. Again, you will see the garbage as the server end of the PPP connection starts up.

If you do not see this immediately after connecting (and logging in and starting the PPP server if required), press Enter to see if this starts the PPP server.

At this point, you can hang up your modem (usually, type +++ quickly and then issue the ATH0 command once your modem responds with OK).

If you can't get your modem to work, read your modem manual, the man pages for your communications software and the Serial HOWTO. Once you have this sorted out, carry on as above.

Using Internet servers with dynamic IP numbers. If you are using dynamic IP numbers (and many service providers will only give you a dynamic IP number unless you pay significantly more for your connection), then you have to recognise the limitations this imposes.

First of all, outbound service requests will work just fine. That is, you can send email using sendmail (provided you have correctly set up sendmail), ftp files from remote sites, finger users on other machines, browse the web etc.

In particular, you can answer email that you have brought down to your machine whilst you are off line. Mail will simply sit in your mail queue until you dial back into your ISP.

However, your machine is not connected to the Internet 24 hours a day, nor does it have the same IP number every time it is connected. So it is impossible for you to receive email directed to your machine, and very difficult to set up a web or ftp server that your friends can access! As far as the Internet is concerned your machine does not exist as a unique, permanently contactable machine as it does not have a unique IP number (remember—other machines will be using the IP number when they are allocated it on dial in).

If you set up a WWW (or any other server), it is totally unknown by any user on the Internet UNLESS they know that your machine is connected AND its actual (current) IP number. There are a number of ways they can get this info, ranging from you ringing them, sending them email to tell them or cunning use of ".plan" files on a shell account at your service provider (assuming that your provider allows shell and finger access).

For most users, this is not a problem—all that most people want is to send and receive email (using your account on your service provider) and make outbound connections to WWW, ftp and other servers on the Internet. If you must have inbound connections to your server, you should really get a static IP number.

PPP connection files. You now need to be logged in as root to create the directories and edit the files needed to set up PPP. PPP uses a number of files to connect and set up a PPP connection. These differ in name and location between PPP 2.1.2 and 2.2.

For PPP 2.1.2 the files are:

```
/usr/sbin/pppd          # the PPP binary
/usr/sbin/ppp-on        # the dialer/connection script
/usr/sbin/ppp-off       # the disconnection script
/etc/ppp/options        # the options pppd uses for all connections
/etc/ppp/options.ttyXX  # the options specific to a connection on this port
```

For PPP 2.2 the files are:

```
/usr/sbin/pppd                  # the PPP binary
/etc/ppp/scripts/ppp-on         # the dialer/connection script
/etc/ppp/scripts/ppp-on-dialer  # part 1 of the dialer script
/etc/ppp/scripts/ppp-off        # the actual chat script itself
/etc/ppp/options                # the options pppd uses for all connections
/etc/ppp/options.ttyXX          # the options specific to a connection on this port
```

Red Hat Linux users should note that the standard Red Hat 4.X installation places these scripts in /usr/doc/ppp-2.2.0f-2/scripts.

In your `/etc` directory there should be a ppp directory:

```
drwxrwxr-x   2 root     root         1024 Oct  9 11:01 ppp
```

If it does not exist—create it with these ownerships and permissions.

If the directory already existed, it should contain a template options file called options.tpl. This file is included below in case it does not.

Print it out as it contains an explanation of nearly all the PPP options (these are useful to read in conjunction with the pppd man pages). Whilst you can use this file as the basis of your /etc/ppp/options file, it is probably better to create your own options file that does not include all the comments in the template - it will be much shorter and easier to read/maintain.

Some distributions of PPP seem to have lost the options.tpl file. You should examine the PPP-HOWTO document for the complete version.

What options should I use? Well, as in all things that depends (sigh). The options specified here should work with most servers.

However, if it does NOT work, READ THE TEMPLATE FILE (/etc/ppp/options.tpl) and the pppd man pages and speak to the sysadmin/user support people who run the server to which you are connecting.

You should also note that the connect scripts presented here also use some command line options to pppd to make things a bit easier to change.

```
# /etc/ppp/options (no PAP/CHAP support)
#
# Prevent pppd from forking into the background -detach
#
# use the modem control lines
modem
# use uucp style locks to ensure exclusive access to the serial device
lock
# use hardware flow control
crtscts
# create a default route for this connection in the routing table
defaultroute
# do NOT set up any "escaped" control sequences
asyncmap 0
# use a maximum transmission packet size of 552 bytes
mtu 552
# use a maximum receive packet size of 552 bytes
mru 552
#
#-------END OF SAMPLE /etc/ppp/options (no PAP/CHAP support)
```

Setting up the PPP connection manually. Now that you have created your /etc/ppp/options and /etc/resolv.conf files (and, if necessary, the /etc/ppp/pap—chap-secrets file), you can test the settings by manually establishing a PPP connection. (Once we have the manual connection working, we will automate the process).

To do this, your communications software must be capable of quitting WITHOUT re-setting the modem. Minicom can do this with the sequence Control-A Q

- Make sure you are logged in as root.
- Fire up you communications software (such as minicom), dial into the PPP server and log in as normal. If you need to issue a command to start up PPP on the server, do so. You will now see the garbage you saw before.
- If you are using PAP or CHAP, then merely connecting to the remote system should start PPP on the remote and you will see the garbage without logging in (although this may not happen for some servers - try pressing Enter and see if the garbage starts up).
- Now quit the communications software without resetting the modem and at the Linux prompt (as root) type

  ```
  # pppd -d /dev/ttyS0 38400 &
  ```

  Substituting the name of the device your modem is connected to, of course.

  The -d option enables debugging—the PPP connection start-up conversation will be logged to your system log—which is useful for tracing problems later.
- Your modem lights should now flash as the PPP connection is established. It will take a short while for the PPP connection to be made.

At this point you can look at the PPP interface, by issuing the command

```
# ifconfig
```

In addition to any Ethernet and loop back devices you have, you should see something like :

```
ppp0      Link encap:Point-Point Protocol
          inet addr:10.144.153.104  P-t-P:10.144.153.51 Mask:255.255.255.0
          UP POINTOPOINT RUNNING  MTU:552  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0
          TX packets:0 errors:0 dropped:0 overruns:0
```

Where

- inet addr:10.144.153.10 is the IP number of your end of the link.
- P-t-P:10.144.153.5 is the SERVER's IP number.

(Ifconfig will not report these IP numbers, but the ones used by your PPP server.) Note: ifconfig also tells you that the link is UP and RUNNING!

You should also be able to see a route to the the remote host (and beyond). To do this, issue the command

```
# route -n
```

You should see something like:

```
Kernel routing table
Destination     Gateway         Genmask         Flags MSS    Window Us
10.144.153.3    *               255.255.255.255 UH    1500   0
127.0.0.0       *               255.0.0.0       U     3584   0       1
10.0.0.0        *               255.0.0.0       U     1500   0       3
default         10.144.153.3    *               UG    1500   0
```

Of particular importance here, notice we have TWO entries pointing to our PPP interface.

The first is a HOST route (indicated by the H flag) and that allows us to see the host to which we are connected to—but no further.

The second is the default route (established by giving pppd the option defaultroute. This is the route that tells our Linux PC to send any packets NOT destined for the local Ethernet(s)—to which we have specific network routes—to the PPP server itself. The PPP server then is responsible for routing our packets out onto the Internet and routing the return packets back to us.

If you do not see a routing table with two entries, something is wrong. In particular if your syslog shows a message telling you pppd is not replacing an existing default route, then you have a default route pointing at your Ethernet interface—which MUST be replaced by a specific network route: YOU CAN ONLY HAVE ONE DEFAULT ROUTE!!!

You will need to explore your system initialization files to find out where this default route is being set up (it will use a route add default... command). Change this command to something like route add net....

Now test the link by 'pinging' the server at its IP number as reported by the ifconfig output, i.e.

```
# ping 10.144.153.51
```

You should receive output like

```
PING 10.144.153.51 (10.144.153.51): 56 data bytes
64 bytes from 10.144.153.51: icmp_seq=0 ttl=255 time=328.3 ms
64 bytes from 10.144.153.51: icmp_seq=1 ttl=255 time=190.5 ms
64 bytes from 10.144.153.51: icmp_seq=2 ttl=255 time=187.5 ms
64 bytes from 10.144.153.51: icmp_seq=3 ttl=255 time=170.7 ms
```

This listing will go on for ever—to stop it press Control-C, at which point you will receive some more information:

```
--- 10.144.153.51 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 170.7/219.2/328.3 ms
```

Now try pinging a host by name (not the name of the PPP server itself) but a host at another site that you KNOW is probably going to be up and running. For example

```
# ping sunsite.unc.edu
```

This time there will be a bit of a pause as Linux obtains the IP number for the fully qualified host name you have 'ping'ed from the DNS you specified in /etc/resolv.conf—so don't worry (but you will see your modem lights flash). Shortly you will receive output like

```
 PING sunsite.unc.edu (152.2.254.81): 56 data bytes
64 bytes from 152.2.254.81: icmp_seq=0 ttl=254 time=190.1 ms
64 bytes from 152.2.254.81: icmp_seq=1 ttl=254 time=180.6 ms
64 bytes from 152.2.254.81: icmp_seq=2 ttl=254 time=169.8 ms
64 bytes from 152.2.254.81: icmp_seq=3 ttl=254 time=170.6 ms
64 bytes from 152.2.254.81: icmp_seq=4 ttl=254 time=170.6 ms
```

Again, stop the output by pressing Control-C and get the statistics...

```
--- sunsite.unc.edu ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 169.8/176.3/190.1 ms
```

If you don't get any response, try pinging the IP address of the DNS server at your ISP's site. If you get a result from this, then it looks like you have a problem with /etc/resolv.conf.

If this doesn't work, you have a routing problem, or your ISP has a problem routing packets back to you. Check your routing table as shown above and if that is OK, contact your ISP. A good test of the ISP is to use another operating system to connect. If you can get beyond your ISP with that, then the problem is at your end.

If everything works, shut down the connection by typing

```
# ppp-off
```

After a short pause, the modem should hang itself up.

If that does not work, either turn off your modem or fire up your communications software and interrupt the modem with +++ and then hang up with ATH0 when you receive the modem's OK prompt.

You may also need to clean up the lock file created by pppd by typing

```
# rm -f /var/lock/LCK..ttyS<x>
```

6.2.3 Creating the connection scripts.

You can continue to log in by hand as shown above, it is much neater to set up some scripts to do this automatically for you.

A set of scripts automates the log in and PPP start up so all you have to do (as root or as a member of the PPP group) is issue a single command to fire up your connection.

If your ISP does NOT require the use of PAP/CHAP, these are the scripts for you.

If the PPP package installed correctly, you should have two example files. For PPP 2.1.2 they are in /usr/sbin and for PPP 2.2 they are in /etc/ppp/scripts. They are called

for PPP-2.1.2

```
ppp-on
ppp-off
```

and for PPP-2.2

```
ppp-off
ppp-on
ppp-on-dialer
```

Now, if you are using PPP 2.1.2, I strongly urge you to delete the sample files. There are potential problems with these—and don't tell me they work fine—I used them for ages too (and recommended them in the first version of this HOWTO)!

For the benefit of PPP 2.1.2 users, here are BETTER template versions, taken from the PPP 2.2 distribution. I suggest you copy and use these scripts instead of the old PPP-2.1.2 scripts.

15.2. The ppp-on script

This is the first of a PAIR of scripts that actually fire up the connection.

```
#!/bin/sh
#
# Script to initiate a PPP connection. This is the first part of the
# pair of scripts. This is not a secure pair of scripts as the codes
# are visible with the 'ps' command. However, it is simple.
#
# These are the parameters. Change as needed.
TELEPHONE=555-1212      # The telephone number for the connection
ACCOUNT=george          # The account name for logon (as in 'George Burns')
PASSWORD=gracie         # The password for this account (and 'Gracie Allen')
LOCAL_IP=0.0.0.0        # Local IP address if known. Dynamic = 0.0.0.0
REMOTE_IP=0.0.0.0       # Remote IP address if desired. Normally 0.0.0.0
NETMASK=255.255.255.0   # The proper netmask if needed
#
# Export them so that they will be available to 'ppp-on-dialer'
export TELEPHONE ACCOUNT PASSWORD
#
# This is the location of the script which dials the phone and logs
# in.  Please use the absolute file name as the $PATH variable is not
# used on the connect option.  (To do so on a 'root' account would be
# a security hole so don't ask.)
#
DIALER_SCRIPT=/etc/ppp/ppp-on-dialer
#
# Initiate the connection
#
#
exec /usr/sbin/pppd debug /dev/ttySx 38400 \
        $LOCAL_IP:$REMOTE_IP \
        connect $DIALER_SCRIPT
```

Here is the ppp-on-dialer script:

```
#!/bin/sh
#
```

```
# This is part 2 of the ppp-on script. It will perform the connection
# protocol for the desired connection.
#
/usr/sbin/chat -v                                       \
        TIMEOUT         3                               \
        ABORT           '\nBUSY\r'                      \
        ABORT           '\nNO ANSWER\r'                 \
        ABORT           '\nRINGING\r\n\r\nRINGING\r'    \
        ''              \rAT                            \
        'OK-+++\c-OK'   ATH0                            \
        TIMEOUT         30                              \
        OK              ATDT$TELEPHONE                  \
        CONNECT         ''                              \
        ogin:--ogin:    $ACCOUNT                        \
        assword:        $PASSWORD
```

For PPP-2.2, the ppp-off script looks like:

```
#!/bin/sh
###############################################################################
#
# Determine the device to be terminated.
#
if [ "$1" = "" ]; then
        DEVICE=ppp0
else
        DEVICE=$1
fi

###############################################################################
#
# If the ppp0 pid file is present then the program is running. Stop it
if [ -r /var/run/$DEVICE.pid ]; then
        kill -INT `cat /var/run/$DEVICE.pid`
#
# If the kill did not work then there is no process running for this
# pid. It may also mean that the lock file will be left. You may wish
# to delete the lock file at the same time.
        if [ ! "$?" = "0" ]; then
                rm -f /var/run/$DEVICE.pid
                echo "ERROR: Removed stale pid file"
```

```
                exit 1
        fi
#
# Success. Let pppd clean up its own junk.
        echo "PPP link to $DEVICE terminated."
        exit 0
fi
#
# The ppp process is not running for ppp0
echo "ERROR: PPP link is not active on $DEVICE"
exit 1
```

6.2.4 Editing the supplied PPP startup scripts.

As the new scripts come in two parts, and we will edit them in turn.

The `ppp-on` script. You will need to edit the ppp-on script to reflect YOUR user name at your ISP, YOUR password at your ISP, the telephone number of your ISP.

Each of the lines like TELEPHONE= actually set up shell variables that contain the information to the right of the '=' (excluding the comments of course). So edit each of these lines so it is correct for your ISP and connection.

Also, as you are setting the IP number (if you need to) in the /etc/ppp/options file, DELETE the line that says:

```
$LOCAL_IP:$REMOTE_IP \
```

Also, make sure that the shell variable DIALER_SCRIPT points at the full path and name of the dialer script that you are actually going to use. So, if you have moved this or renamed the script, make sure you edit this line correctly in the ppp-on script!

The `ppp-on-dialer` script. This is the second of the scripts that actually brings up our ppp link.

Note: a chat script is normally all on one line. the backslashes are used to allow line continuations across several physical lines (for human readability) and do not form part of the script itself.

However, it is very useful to look at it in detail so that we understand what it is actually (supposed) to be doing!

A chat script is a sequence of ”expect string” ”send string” pairs. In particular, note that we ALWAYS expect something before we send something.

If we are to send something WITHOUT receiving anything first, we must use an empty expect string (indicated by ””) and similarly for expecting something without sending anything! Also, if a string consists of several words, (e.g. NO CARRIER), you must quote the string so that it is seen as a single entity by chat.

The chat line in our template is:

```
exec /usr/sbin/chat -v
```

Invoke chat, the -v tells chat to copy ALL its I/O into the system log (usually /var/log/messages). Once you are happy that the chat script is working reliably, edit this line to remove the -v to save unnecessary clutter in your syslog.

```
TIMEOUT         3
```

This sets the timeout for the receipt of expected input to three seconds. You may need to increase this to say 5 or 10 seconds if you are using a really slow modem!

```
ABORT           '\nBUSY\r'
```

If the string BUSY is received, abort the operation.

```
ABORT           '\nNO ANSWER\r'
```

If the string NO ANSWER is received, abort the operation

```
ABORT           '\nRINGING\r\n\r\nRINGING\r'
\begin{verbatim} \end{tscreen}

  If the (repeated) string RINGING is received, abort the operation.
  This is because someone is ringing your phone line!

\begin{tscreen} \begin{verbatim}
  "             \rAT
```

Expect nothing from the modem and send the string AT.

```
OK-+++\c-OK   ATH0
```

This one is a bit more complicated as it uses some of chat's error recovery capabilities.

What is says is...Expect OK, if it is NOT received (because the modem is not in command mode) then send +++ (the standard Hayes-compatible modem string that returns the modem to command mode) and expect OK. Then send ATH0 (the modem hang up string). This allows your script to cope with the situation of your modem being stuck on-line!

```
TIMEOUT         30
```

Set the timeout to 30 seconds for the remainder of the script. If you experience trouble with the chat script aborting due to timeouts, increase this to 45 seconds or more.

```
OK              ATDT$TELEPHONE
```

Expect OK (the modem's response to the ATH0 command) and dial the number we want to call.

```
CONNECT         ''
```

Expect CONNECT (which our modem sends when the remote modem answers) and send nothing in reply.

```
ogin:--ogin:    $ACCOUNT
```

Again, we have some error recovery built in here. Expect the login prompt (...ogin:) but if we don't receive it by the timeout, send a return and then look for the login prompt again. When the prompt is received, send the username (stored in the shell variable $ACCOUNT).

```
assword:        $PASSWORD
```

Expect the password prompt and send our password (again, stored in a shell variable).

This chat script has reasonable error recovery capability. Chat has considerably more features than demonstrated here. For more information consult the chat manual page (man 8 chat).

6.2.5 Starting PPP at the server end.

While the `ppp-on-dialer` script is fine for servers that automatically start pppd at the server end once you have logged in, some servers require that you explicitly start PPP on the server.

If you need to issue a command to start up PPP on the server, you DO need to edit the ppp-on-dialer script.

At the END of the script (after the password line) add an additional expect send pair—this one would look for your login prompt (beware of characters that have a special meaning in the Bourne shell, like

```
$[]
```

Once chat has found the shell prompt, chat must issue the ppp start up command required for your ISPs PPP server.

In one author's case, the PPP server uses the standard Linux Bash prompt

```
[hartr@kepler hartr]$
```

which requires the response

```
# ppp
```

to start up PPP on the server.

It is a good idea to allow for a bit of error recovery here, so use

```
hartr--hartr ppp
```

This says, if we don't receive the prompt within the timeout, send a carriage return and looks for the prompt again.

Once the prompt is received, then send the string ppp.

Note: don't forget to add a to the end of the previous line so chat still thinks the entire chat script is on one line!

Unfortunately, some servers produce a very variable set of prompts! You may need to log in several times using minicom to understand what is going on and pick the stable "expect" strings.

6.2.6 If your PPP server uses PAP (Password Authentication Protocol).

If the server to which you are connecting requires PAP or CHAP authentication, you have a little bit more work.

To the above options file, add the following lines

```
#
# force pppd to use your ISP user name as your 'host name' during the
# authentication process
name <your ISP user name>       # you need to edit this line
```

```
#
# If you are running a PPP *server* and need to force PAP or CHAP
# uncomment the appropriate one of the following lines. Do NOT use
# these is you are a client connecting to a PPP server (even if it uses PAP
# or CHAP) as this tells the SERVER to authenticate itself to your
# machine (which almost certainly can't do---and the link will fail).
#+chap
#+pap
#
# If you are using ENCRYPTED secrets in the /etc/ppp/pap-secrets
# file, then uncomment the following line.
# Note: this is NOT the same as using MS encrypted passwords as can be
# set up in MS RAS on Windows NT.
#+papcrypt
```

6.2.7 Using MSCHAP.

Microsoft Windows NT RAS can be set up to use a variation on CHAP (Challenge/Handshake Authentication Protocol). In your PPP sources you will find a file called README.MSCHAP80 that discusses this. You can determine if the server is requesting authentication using this protocol by enabling debugging for pppd. If the server is requesting MS CHAP authentication, you will see lines like

```
rcvd [LCP ConfReq id=0x2 <asyncmap 0x0> <auth chap 80> <magic 0x46a3>]
```

The critical information here is `auth chap 80.`

In order to use MS CHAP, you will need to recompile pppd to support this. Please see the instructions in the README.MSCHAP80 file in the PPP source file for instructions on how to compile and use this variation.

If you are using pap or chap authentication, then you also need to create the secrets file. These are 1)/etc/ppp/pap-secrets and 2)/etc/ppp/chap-secrets.

They must be owned by user root, group root and have file permissions 740 for security. The first point to note about PAP and CHAP is that they are designed to authenticate computer systems not users. In other words, once your computer has made its PPP connection to the server, ANY user on your system can use that connection—not just you.

PAP can (and for CHAP DOES) require bidirectional authentication—that is a valid name and secret is required on each computer for the other computer involved. However, this is NOT the way most PPP servers offering dial up PPP PAP-authenticated connections operate.

That being said, your ISP will probably have given you a user name and password to allow you to connect to their system and thence the Internet. Your ISP is not interested in your computer's name at all, so you will probably need to use the user name at your ISP as the name for your computer. This is done using the name user name option to pppd. So, if you are to use the user name given you by your ISP, add the line

```
name your_user name_at_your_ISP
```

to your /etc/ppp/options file.

Technically, you should really use `user our_user name_at_your_ISP` for PAP, but pppd is sufficiently intelligent to interpret name as user if it is required to use PAP. The advantage of using the name option is that this is also valid for CHAP.

As PAP is for authenticating computers, technically you need also to specify a remote computer name. However, as most people only have one ISP, you can use a wild card (*) for the remote host name in the secrets file.

The `/etc/ppp/pap-secrets` file looks like

```
# Secrets for authentication using PAP
# client        server       secret     acceptable_local_IP_address
```

The four fields are white space delimited and the last one can be blank (which is what you want for a dynamic and probably static IP allocation from your ISP).

Suppose your ISP gave you a user name of fred and a password of flintstone you would set the name fred option in `/etc/ppp/options` and set up your `/etc/ppp/pap-secrets` file as follows

```
# Secrets for authentication using PAP
# client        server  secret          acceptable local IP address
fred            *       flintstone
```

This says for the local machine name fred (which we have told pppd to use even though it is not our local machine name) and for ANY server, use the password (secret) of flintstone.

Note that we do not need to specify a local IP address, unless we are required to FORCE a particular local, static IP address. Even if you try this, it is unlikely to work as most PPP servers (for security reasons) do not allow the remote system to set the IP number they are to be given.

This requires that you have mutual authentication methods—that is you must allow for both your machine to authenticate the remote server AND the remote server to authenticate your machine.

So, if your machine is fred and the remote is barney, your machine would set name fred remotename barney and the remote machine would set name barney remotename fred in their respective /etc/ppp/options.ttySx files.

The `/etc/chap-secrets` file for fred would look like

```
# Secrets for authentication using CHAP
# client        server  secret            acceptable local IP addresses
fred            barney  flintstone
barney          fred    wilma
```

and for barney

```
# Secrets for authentication using CHAP
# client        server  secret            acceptable local IP addresses
barney          fred    flintstone
fred            barney  wilma
```

Note in particular that both machines must have entries for bidirectional authentication. This allows the local machine to authenticate itself to the remote AND the remote machine to authenticate itself to the local machine.

A chat script for PAP/CHAP authenticated connections. If your ISP is using PAP/CHAP, then your chat script is much simpler. All your chat script needs to do is dial the telephone, wait for a connect and then let pppd handle the logging in!

```
#!/bin/sh
#
# This is part 2 of the ppp-on script. It will perform the connection
# protocol for the desired connection.
#
exec /usr/sbin/chat -v                                  \
        TIMEOUT         3                               \
        ABORT           '\nBUSY\r'                      \
        ABORT           '\nNO ANSWER\r'                 \
        ABORT           '\nRINGING\r\n\r\nRINGING\r'    \
        ''              \rAT                            \
        'OK-+++\c-OK'   ATH0                            \
        TIMEOUT         30                              \
        OK              ATDT$TELEPHONE                  \
        CONNECT         ''                              \
```

As we have already seen, you can turn on debug information logging with the -d option to pppd. The 'debug' option is equivalent to this.

As we are establishing a new connection with a new script, leave in the debug option for now. (Warning: if your disk space is tight, logging pppd exchanges can rapidly extend your syslog file and run you into trouble—but to do this you must fail to connect and keep on trying for quite a few minutes).

Once you are happy that all is working properly, then you can remove this option.

```
exec /usr/sbin/pppd debug file options.myserver /dev/ttyS0 38400 \
```

Testing the connection script. Open a new root Xterm (if you are in X) or open a new virtual console and log in as root.

In this new session, issue the command

```
# tail -f /var/log/messages
```

Many systems log output to `/var/log/messages`. If it has a different name on your system, substitute the name of your system log file in the command above.

In the first window (or virtual console) issue the command

```
# ppp-on &
```

(or whatever name you have called your edited version of `/usr/sbin/ppp-on`). If you do not put the script into the background by specifying & at the end of the command, you will not get your terminal prompt back until ppp exits (when the link terminates).

Now switch back to the window that is tracking your system log.

6.2.8 Shutting down the PPP link.

When you have finished with the PPP link, use the standard `ppp-off` command to shut it down (remember that you must be root or a member of the `ppp` group!).

6.2.9 Troubleshooting common problems once the link is working.

One problem you will find is that many service providers will only support the connection software package that they distribute to new accounts. This is (typically) for Microsoft Windows (—and many service provider help desks seem to know nothing about Unix (or Linux). So, be prepared for limited assistance from them!

You could of course do the individual a favour and educate then about Linux (any ISP help desk person should be reasonably 'with it' in Internet terms and that means they should have a home Linux box—of course it does)!

Address resolution problems. OK—your PPP connection is up and running and you can ping the PPP server by IP number (the second or "remote" IP number shown by ifconfig ppp0), but you can't reach anything beyond this.

First of all, try pinging the IP numbers you have specified in /etc/resolv.conf as name servers. If this works, you can see beyond your PPP server (unless this has the same IP number as the "remote" IP number of your connection). So now try pinging the full Internet name of your service provider

```
ping my.isp.net
```

Substituting, of course, the name of your actual ISP. If this does not work, you have a problem with name resolution. This is probably because of a typo in your `/etc/resolv.conf` file. Check this file carefully against the information in the sample `/etc/resolve.conf` file in section 6.1.1.

If it still doesn't work (and your service provider confirms that his name servers are up and running), you have a problem somewhere else—and check carefully through your Linux installation (looking particularly at file permissions).

If you still can't `ping` your service provider's IP name servers by IP number, either they are down (give them a voice call and check) or there is a routing problem at your service provider's end.

One possibility is that the "remote end" is a Linux PPP server where the IP forwarding option has not been specified in the kernel!

Debugging a failed attempt. There are any number of reasons that your connection does not work— chat has failed to complete correctly, you have a dirty line, etc. So check your syslog for indications.

A very common problem is that people compile PPP support into the kernel and yet when they try to run pppd, the kernel complains that it does not support ppp! There are a variety of reasons this can occur.

- You failed to boot the new kernel that you compiled with PPP support.
- You failed to install the PPP module that you compiled.
- You expected modules to be loaded automatically and they aren't.

- You are using the incorrect version of PPP for your kernel.
- You are not running pppd as root.
- You mistyped something in your startup scripts.
- You are not correctly logging into the server.
- You are not starting PPP on the server.
- The remote PPP process is slow to start.
- Default route not set.

And a host of others. Look in the PPP FAQ (which is really a series of questions and answers). This is a very comprehensive document and the answers are there! If the answer to your problems is not there, the problem is not ppp's fault!

Getting help when totally stuck. If you can't get your PPP link to work, go back through this document and check everything—in conjunction with the output created by "chat-v..." and "pppd -d" in you system log.

Also consult the PPP documentation and FAQ plus the other documents mention herein!

If you are still stuck, try the comp.os.linux.misc and comp.os.linux.networking newsgroups are reasonably regularly scanned by people that can help you with PPP as is comp.protocols.ppp

If you do choose to seek help in the USENET newsgroups, please don't post a very long message consisting of debugging output. This wastes huge amounts of network bandwidth. It is much better to describe the problem and perhaps include a few lines of debugging output (definitely no more than one screenful).

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 messages, 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 6.5 and 6.6) 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 HOWTO, 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 A for more information.

6.4 Networking with Microsoft Systems.

Samba is a suite of programs which work together to allow clients to access to a server's filespace and printers via the SMB (Server Message Block) protocol. Initially written for Unix, Samba now also runs on Netware, OS/2 and VMS.

In practice, this means that you can redirect disks and printers to Linux disks and printers from Lan Manager clients, Windows for Workgroups 3.11 clients, Windows NT clients, Linux clients and OS/2 clients. This gives the capability for these operating systems to behave much like a LAN Server or Windows NT Server machine, only with added functionality and flexibility designed to make life easier for administrators.

Samba: Integrating UNIX and Windows contains complete information on configuring and using Samba under Linux. The Samba home pages are located at http://samba.anu.edu.au/samba/ and the SMB HOWTO should also be of help.

6.5 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 care 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 `sendmail`. It is able to send both local and remote TCP/IP and UUCP e-mail. An alternative to `sendmail` is `Smail`.

The Linux Mail HOWTO 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 6.1 and 6.3). The UUCP and TCP/IP documents listed in Appendix A 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`.

6.6 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 "news groups" 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 news groups and handles delivering articles to other machines (if you are on a network). The news client, or *news reader*, 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 news readers, as well, such as `rn` and `tin`. The choice of news reader is more or less a matter of taste; all news readers should work equally well with different versions of the server software. That is, the news reader 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 news reader for the users. The news server will store the articles in a directory such as `/var/spool/news`, and the news reader 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 news reader 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 news reader 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 news reader 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 news reader. (Most news readers 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 news reader 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 6.1).

- 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 news reader, 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.

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 News HOWTO from `sunsite.unc.edu` in `/pub/Linux/docs/HOWTO`. 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 docu-

ment “How to become a Usenet site,” available from `ftp.uu.net`, in the directory `/usenet/news.announce.newusers`.

Appendix A

Sources of Linux Information

This appendix contains information on various sources of Linux information, such as online documents, books, and more. Many of these documents are available either in printed form, or electronically from the Internet or BBS systems. Many Linux distributions also include much of this documentation in the distribution itself, so after you have installed Linux these files may be present on your system.

Online documents.

These documents should be available on any of the Linux FTP archive sites (see Appendix B for a list). If you do not have direct access to FTP, you may be able to locate these documents on other online services (such as CompuServe, local BBSs, and so on). If you have access to Internet mail, you can use the `ftpmail` service to receive these documents. See Appendix B for more information.

In particular, the following documents may be found on `sunsite.unc.edu` in the directory `/pub/Linux/docs`. Many sites mirror this directory; however, if you're unable to locate a mirror site near you, this is a good one to fall back on.

The Linux Frequently Asked Questions with Answers

The Linux Frequently Asked Questions list, or "FAQ", is a list of common questions (and answers!) about Linux. This document is meant to provide a general source of information about Linux, common problems and solutions, and a list of other sources of information. Every new Linux user should read this document. It is available in a number of for-

mats, including plain ASCII, PostScript, and HTML. The Linux FAQ is maintained by Roert Kiesling, `kiesling@terracom.net`.

The Linux META-FAQ

The META-FAQ is a collection of "metaquestions" about Linux; that is, sources of information about the Linux system, and other general topics. It is a good starting place for the Internet user wishing to find more information about the system. It is maintained by Michael K. Johnson, `johnsonm@sunsite.unc.edu`.

The Linux INFO-SHEET

The Linux INFO-SHEET is a technical introduction to the Linux system. It gives an overview of the system's features and available software, and also provides a list of other sources of Linux information. The format and content is similar in nature to the META-FAQ; incidentally, it is also maintained by Michael K. Johnson.

Linux Journal *Linux Journal* makes selected articles from the magazine (published monthly) available in electronic form from their web site, `http://www.linuxjournal.com/`. Articles cover topics for beginning to advanced users, include features about Linux being used in the "real world", and the frequently referenced Linux distribution comparison articles and tables.

Linux Gazette A free on-line publication found at `http://www.linuxgazette.com/`, *Linux Gazette* offers answers and entertainment, "making Linux just a little more fun." The Gazette is produced by SSC, the publishers of *Linux Journal*. Contact gazette@ssc.com for more information.

Linux Resources

The Linux Resources (`http://www.linuxresources.com/`) cover "What it is, where to get it, how to find all the information you need to get it running and much more. What it is, where to get it, how to find all the information you need to get it running and much more."

The Linux Software Map

The Linux Software Map is a list of many applications available for

Linux, where to get them, who maintains them, and so forth. It is far from complete—to compile a complete list of Linux software would be nearly impossible. However, it does include many of the most popular Linux software packages. If you can't find a particular application to suit your needs, the LSM is a good place to start. It is maintained by Lars Wirzenius, `lars.wirzenius@helsinki.fi`.

The Linux HOWTO Index

The Linux HOWTOs are a collection of "how to" documents, each describing in detail a certain aspect of the Linux system. They are maintained by Matt Welsh, `mdw@sunsite.unc.edu`. The `HOWTO-Index` lists the HOWTO documents which are available.

Other online documents

If you browse the `docs` subdirectory of any Linux FTP site, you'll see many other documents which are not listed here: A slew of FAQ's, interesting tidbits, and other important information. This miscellany is difficult to categorize here; if you don't see what you're looking for on the list above, just take a look at one of the Linux archive sites listed in Appendix B.

Intorduction to the Linux Documentation Project.

The Linux Documentation Project (LDP) is working on developing good, reliable documentation for the Linux operating system. The overall goal of the LDP authors is to write documents in various formats that cover installing, configuring, and using Linux. The LDP produces documents in a variety of formats: plain text that you can read anywhere, HTML documents you can read with a browser, man pages that can be read online or in a book, and typeset documentation that can be printed and read in books.

The LDP's "home" is its web page, found at `http://sunsite.unc.edu/LDP/` and countless mirrors listed at `http://sunsite.unc.edu/LDP/hmirrors.html`. This is the place to check for updates, news, and some documents that only exist online. A few documents that exist only online are;

- *Linux Gazette*, a monthly collection of unedited articles and letters from Linux users everywhere.

- *The Linux Kernel Hackers' Guide*, an interactive, edited forum where Linux kernel developers talk about kernel development issues.
- Special HOWTOs, HOWTO documents that rely on things that cannot be supported in plain text versions.

In addition to the LDP web pages, there are four basic types of documentation produced by the LDP: Guides, HOWTOs and mini-HOWTOs, man pages, and FAQs.

- Guides
  Entire books on complex topics.
- HOWTOs and mini-HOWTOs
  Documents with full coverage of a fairly well-defined topic or simple coverage, usually of a single task.
- man pages
  Documentation for single programs, file formats, and library functions in standard UNIX reference format.
- FAQs
  Frequently Asked Questions on verious topics, including the Linux FAQ.

If you have comments about any particular document in this set, feel free to send it to the author. All documents have the author's email address to send comments to, and while the authors may not always have time to respond, they do read and consider thoughtful comments on their work. Your comments help make the next versions of these documents better. If you have comments or questions about the LDP in general, please contact Greg Hankins via email at <`gregh@sunsite.unc.edu`>.

Books and other published works.

Linux Journal is a monthly magazine for and about the Linux community, written and produced by a number of Linux developers and enthusiasts. It is distributed worldwide, and is an excellent way to keep in touch with the dynamics of the Linux world, especially if you don't have access to USENET news.

At the time of this writing, subscriptions to *Linux Journal* are US$22/year in the United States, US$27 in Canada, and US$37 elsewhere. To subscribe, or for more information, write to Linux Journal, PO Box 55549, Seattle, WA, 98155-0549, USA, or

call +1 206 782-7733, or toll free 1-888-66-Linux in North America. Their FAX number is +1 206 782-7191, and e-mail address is `linux@ssc.com`. You can also find a *Linux Journal* FAQ and sample articles via anonymous FTP on `sunsite.unc.edu` in `/pub/Linux/docs/linux-journal`.

As we have said, not many books have been published dealing with Linux specifically. However, if you are new to the world of UNIX, or want more information than is presented here, we suggest that you take a look at the following books which are available.

Linux Titles.

Title: *The Complete Linux Kit*
Author: Stefan Strobel, Rainer Maurer, Stefan Middendorf, Volker Elling
Publisher: Springer Verlag, 1997
ISBN: 0387142371, $59.95

> Publisher desription: This two-volume, four-CD-ROM bundle consists of *Linux Universe: Installation and Configuration* and *Linux: Unleashing the Workstation in Your PC*. The former is a book/CD-ROM package which includes a full installable version of Linux 2.0 and a detailed installation guide for that version. The latter is a highly detailed guide to installing and administering any Linux system along with a host of Linux tools and applications.

Title: *Linux: Installation, Configuration, and Use*
Author: Michael Kofler
Publisher: Addison-Wesley, 1997
ISBN: 0201178095, $34.95

> Publisher Description: A comprehensive and practical guide, this book covers the installation, configuration and use of Linux. Michael Kofler walks readers through installation to simple administration and the use of Emacs editor, LaTeX typesetting, and the Tcl/Tk programming language. The CD-ROM contains RedHat Linux 4.1 and complete kernel sources for versions 2.0.29 and 2.1.28.

Title: *Linux: Configuration and Installation (3rd Edition)*
Author: Patrick Volkerding
Publisher: IDG Books, 1997
ISBN: 1558285660, $39.95

Summary: Our 2-CD-ROM pack offers one of the most popular Linux distributions, Slackware 96, and comes directly from Patrick Volkerding, the creator of Salckware. Provides you with undocumented tips and techniques for setting up, using, and optimizing your Linux system.

Title: *Linux in Plain English*
Author: Patrick Volkerding, Kevin Reichard
Publisher: IDG Books, 1997
ISBN: 1558285423 , $19.95

Includes detailed listings of all Linux commands, covering the GNU command set and the Linux Bash Shell, file manipulation, text processing, printing, the Internet, and FTP and system administration.

Title: *Linux for Dummies (1st Ed)*
Author: Craig Witherspoon, Coletta Witherspoon
Publisher: IDG Books, 1998
ISBN: 0764502751 , $24.99

A beginner's book on Linux

Title: *Linux for Dummies Quick Reference*
Author: Phil Hughes
Publisher: IDG Books, 1998
ISBN: 0764503022, $14.99

Publisher Description: This ”Quick Reference” is a handy guide to the most commonly used Linux commands and tasks, emphasizing the most popular text editors, Windows interfaces, and Linux flavors. The book covers the shell commands, basic shell scripting commands, and common networking and administration commands.

Title: *Discover Linux (1st Ed)*
Author: Steve Oualline
Publisher: IDG Books, 1997
ISBN: 0764531050 , $24.99

Publisher Description: Users who know UNIX and want to learn and use Linux will find what they need to know in this title. Different types of audiences include programmers, network administrators, people who need an easy Internet/Web connection, users who need a secure firewall machine, and game players. The CD-ROM includes the popular and easy-to-install RedHat Linux 4.1 distribution.

Title: *Complete Red Hat Linux Resource Kit C/Dos/Us*
Author: Collective Work
Publisher: Macmillan Digital, 1997
ISBN: 0672310570 , $64.99

Publisher Description: Turn your PC into a powerful UNIX workstation! Red Hat Linux is the hottest implementation of the Linux operating system. This collection includes the software’s latest version, plus everything else you need to run the popular 32-bit UNIX operating system. Features the easiest installation of any Linux system, more than 180 Linux programs, and an RPM manager allowing new version updates without system reinstallation. Also comes with Apache web server, games and 250-page user guide.

Title: *Running Linux (2nd Edition)*
Author: Matt Welsh, Lar Kaufman

Publisher: O'Reilly and Associates, 1996
ISBN: 1565921518 , $29.95

Publisher Description: This second edition of Running Linux covers everything you need to understand, install, and start using the Linux operating system. It includes a comprehensive installation tutorial, complete information on system maintenance, tools for document development and programming.

Title: *Linux: Installation, Configuration, and Use*
Author: Michael Kofler
Publisher: Addison-Wesley Pub Co, 1997
ISBN: 0201178095 , $34.95

Publisher Description: A comprehensive and practical guide, this book covers the installation, configuration and use of Linux. Michael Kofler walks readers through installation to simple administration and the use of Emacs editor, LaTeX typesetting, and the Tcl/Tk programming language. The CD-ROM contains RedHat Linux 4.1 and complete kernel sources for versions 2.0.29 and 2.1.28.

Title: *The No B.S. Guide to Linux*
Author: Bob Rankin
Publisher: No Starch Press, 1997
ISBN: 1886411042 , $34.95

Publisher Description: This guide provides all the information new users need without burying them in history lessons and technical details. The question-and-answer format of this guide lets readers troubleshoot problems, discover taskbar tricks and shortcuts, and make the transition to a new system as smooth as possible.

Title: *Linux Start-Up Guide : A Self-Contained Introduction*

Author: Fred Hantelmann, A. Faber (Translator)
Publisher: Springer Verlag, 1997
ISBN: 354062676X , $28.00

Publisher Description: This systematic overview for beginners, system administrators, and new users of Linux gives the full details of operating system architecture, basic Linux commands, and typical development and application packages.

Using UNIX.

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

This is a book about the `vi` editor, a powerful text editor found on every UNIX system in the world. It's often important to know and be able to use `vi`, because you won't always have access to a "real" editor such as Emacs.

Title: *VI Tutorial*
Author: Belinda Frazier

Publisher: Specialized Systems Consultants
ISBN: 0-916151-54-9, $6.00

This tutorial provides explanations of examples of vi commands. While some long-time vi users have reported they learned something new the first time they read the VI Tutorial, it is geared towards beginning and intermediate vi users.

Title: *Bourne Shell Tutorial*
Author: Phil Hughes
Publisher: Specialized Systems Consultants
ISBN: 0-916151-39-5, $6.00

This tutorial explains many of the capabilities of the standard UNIX System V shell, commonly called the Bourne Shell. Included is a 4-page reference guide of the commands built into the shell and a 3-page summary of some of the commonly used UNIX commands.

System Administration.

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: *Samba: Integrating UNIX and Windows*

Author: John D. Blair
Publisher: Specialized Systems Consultants
ISBN: 1-57831-006-7, $29.95

Samba is the tool of choice for providing Windows file sharing and printer services from UNIX and UNIX-like systems. Freely available under the GNU Public License, Samba allows UNIX machines to be seamlessly integrated into a Windows network without installing any additional software on the Windows machines. Used in tandem with Linux or FreeBSD, Samba provides a low-cost alternative to the Windows NT Server.

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-HOWTO 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.

The X Window System.

Title: *The X Window System: A User's Guide*
Author: Niall Mansfield
Publisher: Addison-Wesley
ISBN: 0-201-51341-2, ??

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 at first sight.

Programming.

Title: *The C Programming Language*
Author: Brian Kernighan and Dennis Ritchie
Publisher: Prentice-Hall, 1988
ISBN: 0-13-110362-8, $25.00

This book is a must-have for anyone wishing to do C programming on a UNIX system. (Or any system, for that matter.) While this book is not obstensibly UNIX-specific, it is quite applicable to programming C under UNIX.

Title: *The Unix Programming Environment*
Author: Brian Kernighan and Bob Pike
Publisher: Prentice-Hall, 1984
ISBN: 0-13-937681-X, $40.00

An overview to programming under the UNIX system. Covers all of the tools of the trade; a good read to get acquainted with the somewhat amorphous UNIX programming world.

Title: *Advanced Programming in the UNIX Environment*

Author: W. Richard Stevens
Publisher: Addison-Wesley
ISBN: 0-201-56317-7, $50.00

This mighty tome contains everything that you need to know to program UNIX at the system level—file I/O, process control, interprocess communication, signals, terminal I/O, the works. This book focuses on various UNI standards, including POSIX.1, which Linux mostly adheres to.

Kernel hacking.

Title: *Inside Linux: A Look at Operating System Development*
Author: Randolph Bentson
Publisher: Specialized Systems Consultants
ISBN: 0-916151-89-1, $22.00

This book provides an informal introduction to a number of operating system issues by looking at the history of operating systems, by looking at how they are used, and by looking at the details of one operating system. The contents are a conscious effort to braid discussion of history, theory and practice so that the reader can see what goes on inside the system.

Title: *The Design of the UNIX Operating System*
Author: Maurice J. Bach
Publisher: Prentice-Hall, 1986
ISBN: 0-13-201799-7, $70.00

This book covers the algorithms and internals of the UNIX kernel. It is not specific to any particular kernel, although it does lean towards System V-isms. This is the best place to start if you want to understand the inner tickings of the Linux system.

Title: *The Magic Garden Explained*

Author: Berny Goodheart and James Cox
Publisher: Prentice-Hall, 1994
ISBN: 0-13-098138-9, $53.00

This book describes the System V R4 kernel in detail. Unlike Bach's book, which concentrates heavily on the algorithms which make the kernel tick, this book presents the SVR4 implementation on a more technical level. Although Linux and SVR4 are distant cousins, this book can give you much insight into the workings of an actual UNIX kernel implementation. This is also a very modern book on the UNIX kernel—published in 1994.

Title: *Linux Kernel Internals*
Author: Michael Beck
Publisher: Addison-Wesley, 1997
ISBN: 0201331438, $41.95

A look at the code and technical details of the Linux Kernel

Appendix B

FTP Tutorial and Site List

The File Transfer Protocol (FTP) is the set of programs 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.

At the end of this appendix there is a listing of FTP sites where Linux software can be found. Also, if you don't have direct Internet access but are able to exchange electronic mail with the Internet, information on using the `ftpmail` service is included below.

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`) with 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.

Starting `ftp`

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, 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
```

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, 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.

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 file name 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

`cdup`

to change to the parent directory[1] .

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
```

[1] The directory above the current one.

```
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> cd pub
ftp> dir
200 PORT command successful.
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).

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 file names 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 re-

mote 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 file name is the same as the remote one. However, if for example you're downloading the file `README`, and you already have a `README` 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>
```

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
```

Using `ftpmail`

`ftpmail` is a service which allows you to obtain files from FTP archive sites via Internet electronic mail. If you don't have direct Internet access, but are able to send mail to the Internet (from a service such as CompuServe, for example), `ftpmail` is a good way to get files from FTP archive sites. Unfortunately, `ftpmail` can be slow, especially when sending large jobs. Before attempting to download large amounts of software using `ftpmail`, be sure that your mail spool will be able to handle the incoming traffic. Many systems keep quotas on incoming electronic mail, and may delete your account if your mail exceeds this quota. Just use common sense.

`sunsite.unc.edu`, one of the major Linux FTP archive sites, is home to an `ftpmail` server. To use this service, send electronic mail to

```
ftpmail@sunsite.unc.edu
```

with a message body containing only the word:

```
help
```

This will send you back a list of `ftpmail` commands and a brief tutorial on using the system.

For example, to get a listing of Linux files found on `sunsite.unc.edu`, send mail to the above address containing the text

```
open sunsite.unc.edu
cd /pub/Linux
dir
quit
```

You may use the `ftpmail` service to connect to any FTP archive site; you are not limited to `sunsite.unc.edu`. The next section lists a number of Linux FTP archives.

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.ibr.cs.tu-bs.de	134.169.34.15	/pub/os/linux
ftp.denet.dk	129.142.6.74	/pub/OS/linux

Table B.1: Linux FTP Sites

Linux FTP Site List

The table on page 317 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.

`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 that is geographically closest to you.

Appendix C

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

Copyright ©1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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.

GNU GENERAL PUBLIC LICENSE
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 contri-

butions 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

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.