

Slackware Linux Essentials

The Official Guide To Slackware Linux

David Cantrell

Logan Johnson

Chris Lumens

This documentation is licensed under the terms of the GNU General Public License. A copy of this license can be found in [Appendix A](#).

Linux is a registered trademark of Linus Torvalds. Slackware is a registered trademark of BSDi and Patrick Volkerding.

Table of Contents

[Preface](#)

[Conventions Used in This Book](#)

I. [Introduction](#)

1. [An Introduction to Slackware Linux](#)

[What is Linux?](#)

[What is Slackware?](#)

[Open Source and Free Software](#)

2. [Help](#)

[System Help](#)

[Online Help](#)

II. [Installation](#)

3. [Installation](#)

[Getting Slackware](#)

[System Requirements](#)

[Summary](#)

III. [Configuration](#)

4. [System Configuration](#)

[System Overview](#)

[Selecting A Kernel](#)

[Summary](#)

5. [Network Configuration](#)

[Network Hardware](#)

[Network Utilities](#)

[The /etc files](#)

[rc.inet1](#)

[rc.inet2](#)

[NFS \(Network File System\)](#)

[tcp_wrappers](#)

[Summary](#)

6. [The X Window System](#)

[xf86config](#)

[XF86Setup](#)

[Session Configuration Files](#)

[Servers and Window Managers](#)

[Selecting a Desktop](#)

[Exporting displays](#)

[Summary](#)

7. [Bootting](#)

[LILO](#)

[LOADLIN](#)

[Dual Booting](#)

[Summary](#)

IV. [Using Slackware Linux](#)

8. [The Shell](#)

[Users](#)

[The Command Line](#)

[The Bourne Again Shell \(bash\)](#)

[Virtual Terminals](#)

[Summary](#)

9. [Filesystem Structure](#)

[Ownership](#)

[Permissions](#)

[Links](#)

[Mounting Devices](#)

[NFS Mounts](#)

[Summary](#)

10. [Handling Files and Directories](#)

[ls](#)

[cd](#)

[more](#)

[less](#)

[cat](#)

[touch](#)

[echo](#)

[mkdir](#)

[ln](#)

[cp](#)

[mv](#)

[rm](#)

[rmdir](#)

[Summary](#)

11. [Process Control](#)

[Backgrounding](#)

[Foregrounding](#)

[ps](#)

[kill](#)

[top](#)

[Summary](#)

12. [Essential System Administration](#)

[Users and Groups](#)

[Shutting Down Properly](#)

[Summary](#)

13. [Basic Network Commands](#)

[ping](#)

[finger](#)

[telnet](#)

[FTP Clients](#)

[email](#)

[lynx](#)

[wget](#)

[traceroute](#)

[Talking to Other People](#)

[Summary](#)

14. [Archive Files](#)

[gzip](#)

[bzip2](#)

[tar](#)

[zip](#)

[Summary](#)

15. [vi](#)

[Starting vi](#)

[Modes](#)

[Opening Files](#)

[Saving Files](#)

[Quitting vi](#)

[vi Configuration](#)

[vi Keys](#)

[Summary](#)

16. [Slackware Package Management](#)

[Overview of Package Format](#)

[Package Utilities](#)

[Making Packages](#)

[Making Tags and Tagfiles \(for setup\)](#)

[Summary](#)

17. [ZipSlack and BigSlack](#)

[What is ZipSlack/BigSlack?](#)

[Getting ZipSlack/BigSlack](#)

[Installation](#)

[Booting ZipSlack/BigSlack](#)

[Adding, Removing, and Upgrading Software](#)

[Common Problems](#)

[Getting Help](#)

[Summary](#)

[Glossary](#)

A. [The GNU General Public License](#)

[Preamble](#)

[TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND
MODIFICATION](#)

[How to Apply These Terms to Your New Programs](#)

[Next](#)

Preface

Preface

The Slackware Linux operating system is a powerful platform for Intel-based computers. It is designed to be stable, secure, and functional as both a high-end server and powerful workstation.

This book is designed to get you started with the Slackware Linux operating system. It's not meant to cover every single aspect of the distribution, but rather to show what it's capable of and give you a basic working knowledge of the system.

As you gain experience with Slackware Linux, we hope you find this book to be a handy reference. We also hope you'll lend it to all of your friends when they come asking about that cool Slackware Linux operating system you're running .

While this book may not an edge-of-your-seat novel, we certainly tried to make it as entertaining as possible. With any luck, we'll get a movie deal. Of course, we also hope you are able to learn from it and find it useful.

And now, on with the show.

Conventions Used in This Book

This book is written in SGML using the DocBook 4.0 DTD. As such, we used the builtin DocBook elements for filename reference, command reference, and file contents reference. This provides consistent typefaces for all aspects of the book. You'll need to be familiar with a few of our conventions before you continue.

Whenever we mention a command that you are to run, it will look like this:

```
command
```

On rare occasion, a command might be longer than the space on one row of this book. When that happens, we'll wrap the command onto the next line and use a backslash to indicate that the command continues. Here's an example from later in the book:

```
ifconfig eth0 192.168.1.10 broadcast 192.168.1.255 \  
netmask 255.255.255.0
```

Filenames and directories are referred to throughout the book. They will look like this:

```
filename
```

```
directory
```

Screens of command output and the contents of configuration files are also used in the

book. They will appear in this typeface:

```
command output
```

Sometimes when we list commands that you're to run, we will display them as being run from a sample prompt. When a command is meant to be run as a regular user, we will display it on a prompt that is a dollar sign (\$). When a command is meant to be run as root, we will display it on a prompt that is a hash mark (#).

[Prev](#)

Slackware Linux Essentials

[Home](#)

[Next](#)

Introduction

I. Introduction

Table of Contents

1. [An Introduction to Slackware Linux](#)
2. [Help](#)

Chapter 1. An Introduction to Slackware Linux

Table of Contents

[What is Linux?](#)

[What is Slackware?](#)

[Open Source and Free Software](#)

What is Linux?

Linux was started by Linus Torvalds in 1991 as a personal project. He was looking for a way to run a Unix-based operating system without spending a lot of money. In addition, he wanted to learn the ins-and-outs of the 386 processor. It was released free-of-charge to the public so that anyone could hack on it and make improvements under the GNU General Public License (see [the section called *Open Source and Free Software*](#) and [Appendix A](#)).

Today, Linux has grown into a major player in the operating system market. It has been ported to run on a variety of architectures including Compaq's Alpha, Sun's SPARC and UltraSPARC, and Motorola's PowerPC chips (through Apple Macintosh and IBM RS/6000 computers, for example). Linux is now being developed by hundreds (if not thousands) of programmers from all over the world. It runs programs like Sendmail, Apache, and BIND, which is some of the most popular server software on the Internet.

The term Linux really only refers to the kernel - the core of the operating system. This part is responsible for controlling your processor, memory, hard drives, and peripherals. That's all that Linux really does. It controls the operations of your computer and makes sure that all the programs behave. All those programs that make Linux useful are developed by independant groups. The kernel and programs are bundled together by various companies and individuals to make an operating system. We call this a Linux distribution.

What is Slackware?

Slackware was the first Linux distribution to achieve widespread use. It was started by Patrick Volkerding in late 1992. He had gotten introduced to Linux when he needed an inexpensive LISP interpreter for a project. At that time, there were very few distributions, so Patrick went with the distribution from Soft Landing Systems (SLS Linux).

However, SLS had some problems, so Patrick started to fix little bugs as he found them. Eventually, he decided to merge all of those fixes into his own distribution for himself and friends. This private distribution quickly gained popularity, and Patrick made it available to the public under the name of Slackware.

Along the way, Patrick added new things to the distribution like a user-friendly installation program based on a menuing system and the concept of package management. This allows users to easily add, remove, or upgrade software packages from their system.

Open Source and Free Software

Within the Linux community, there are two major ideological movements at work. The Free Software movement, which we'll get into in a moment, is working toward the goal of making all software free of intellectual property restrictions, which it believes hamper technical improvement and work against the good of the community. The Open Source movement is working toward most of the same goals, but takes a more pragmatic approach to them, preferring to base its arguments on the economic and technical merits of making source code freely available, rather than the moral and ethical principles that drive the Free Software Movement.

The Free Software movement is headed up by the Free Software Foundation, which is a fund-raising organization for the GNU project. Free software is more of an ideology. The oft-used expression is free speech, not free beer. In essence, free software is an attempt to guarantee certain rights for both users and developers. These freedoms include the freedom to run the program for any reason, the freedom to study and modify the source code, the freedom to redistribute the source, and the freedom to share any modifications you make. In order to guarantee these freedoms, the GNU General Public License (GPL) was created. The GPL, in brief, provides that anyone distributing a compiled program which is licensed under the GPL must also provide source code, and is free to make modifications to the program as long as those modifications are also made available in source code form. This guarantees that once a program is opened to the community, it cannot be closed except by consent of every author of every piece of code (even the modifications) within it. Most Linux programs are licensed under the GPL.

It is important to note that the GPL does not say anything about price. As odd as it may sound, you can charge for free software. The free part is in the liberties you have with the source code, not in the price you pay for the software. (However, once someone has sold you, or even given you, a compiled program licensed under the GPL they *are* obligated to provide its source code as well.)

At the forefront of the younger Open Source movement, the Open Source Initiative is an organization that solely exists to gain support for open source software. That is, software that has the source code available as well as the ready-to-run program. They do not offer a specific license, but instead they support the various types of open source licenses available.

The idea behind the OSI is to get more companies behind open source by allowing them to write their own open source licenses and have those licenses certified by the Open Source Initiative. Many companies want to release source code, but do not want to use the GPL. Since they cannot radically change the GPL, they are offered the opportunity to provide their own license and have it certified by this organization.

While the Free Software Foundation and the Open Source Initiative work to help each

other, they are not the same thing. The Free Software Foundation uses a specific license and provides software under that license. The Open Source Initiative seeks support for all open source licenses, including the one from the Free Software Foundation. The grounds on which each argues for making source code freely available sometimes divides the two movements, but the very fact that two ideologically diverse groups are working toward the same goal lends credence to the efforts of each.

[Prev](#)

What is Slackware?

[Home](#)[Up](#)[Next](#)

Help

Chapter 2. Help

Table of Contents

[System Help](#)

[Online Help](#)

There are times when you may need help with a specific command, setting up a program, or getting a piece of hardware to work. Luckily, there are a variety of ways that you can get help. If you installed the packages from the F software series, you have a wealth of help already installed. Programs also come with help about their options, configuration files, and usage. Finally, you can check the official Slackware website for help.

System Help

man

man (short for `manual`) is a traditional form of online documentation in Unix and Linux operating systems. Specially formatted files, `man` pages, are written for most commands and distributed with the software. Running **man somecommand** will display the `man` page for (naturally) the command or program **somecommand**.

Because there are so many of them, `man` pages are grouped into enumerated sections. This system has been around so long that you will often see commands, programs, and even programming library functions referred to with their `man` section number. For instance, you might see **man(1)**. This tells you that **man** is documented in section 1 (user commands); you can specify that you want the section 1 `man` page for `man` with the command **man 1 man**. Specifying the section that `man` should look in is useful in the case of multiple items with the same name.

Table 2-1. Man Page Sections

Section	Contents
Section 1	user commands (intro only)
Section 2	system calls
Section 3	C library calls
Section 4	devices (e.g., <code>hd</code> , <code>sd</code>)
Section 5	file formats and protocols (e.g., <code>wtmp</code> , <code>/etc/passwd</code> , <code>nfs</code>)
Section 6	games (intro only)
Section 7	conventions, macro packages, etc. (e.g., <code>nroff</code> , <code>ascii</code>)

In addition to **man**(1), there are the commands **whatis**(1) and **apropos**(1), whose shared purpose is to make it easier to find information in the man system. **whatis** gives a very brief description of system commands, somewhat in the style of a pocket command reference. **apropos** is used to search for a man page containing a given keyword.

See their man pages for details. ;)

The /usr/doc Directory

The source for most packages that we build comes with some sort of documentation. README files, usage instructions, license files... any sort of documentation that comes with the source is included and installed on your system in the /usr/doc directory.

If man pages don't provide enough information, /usr/doc should be your next stop.

HOWTOs and mini-HOWTOs

It is the true spirit of community that brings you the HOWTO/mini-HOWTO collection. These files are exactly what they sound like-- documents describing how to do stuff. If you install the HOWTO collection package, HOWTOs will be installed to /usr/doc/Linux-HOWTOs and the mini-HOWTOs to /usr/doc/Linux-mini-HOWTOs.

Also included in the same package is a collection of FAQs (Frequently Asked Questions lists-- with answers) which are installed to the same place.

These files are well worth reading whenever you're not quite sure how to proceed with something. An amazing range of topics are covered in sometimes surprising detail.

[Prev](#)

Open Source and Free
Software

[Home](#)

[Up](#)

[Next](#)

Online Help

Online Help

In addition to the documentation provided and installable with the Slackware Linux operating system, there are several online resources available.

Website and Forum

www.slackware.com

The official Slackware Linux website is chock full of yummy Slackware documentation. There are introductory help pages, an installation guide, Frequently Asked Questions lists, and all sorts of other good stuff for new (and often experienced) users.

Also available at the website is the Slackware Forum, a section where users can discuss their Slackware experiences and help each other with questions and problems. It's proven to be a very popular and helpful resource, and should probably be your first stop for support. (Because of the wide audience that your message will reach, you may sometimes have a better chance of getting a quick solution; chances are someone's overcome almost any problem you might run into). Please do search the forum for your question, to see if it has already been asked and answered, before posting.

E-mail Support

Anyone who buys an official CD set is entitled to free installation support via e-mail. That said, we're of the old school. We do our best to help anyone who emails us with support questions. Please check your documentation and the website (especially the FAQs and Forum) before e-mailing; you may get a faster answer that way, and the less e-mail we have to answer, obviously the sooner we will be able to help everyone.

The e-mail address for technical support is: <support@slackware.com>. Other e-mail addresses and contact information are listed on the website.

II. Installation

Table of Contents

3. [Installation](#)

Chapter 3. Installation

Table of Contents

[Getting Slackware](#)

[System Requirements](#)

[Summary](#)

Before you can use Slackware Linux, you'll have to obtain and install it. Getting Slackware is as easy as purchasing it or downloading it for free over the internet. Installing it is also easy as long as you have some basic knowledge about your computer and are willing to learn a few other things. The installation program itself is very much a step-by-step process. Because of this, you can be up and running very quickly.

Getting Slackware

The Official Disc and Box Sets

The official Slackware Linux CD set is available from Slackware, Inc. By purchasing the official disc set, you get the convenience of a CD installation, installation support via email, a 30-page installation booklet, and more. The Slackware box set includes the CD set, plus the official Slackware Linux manual. Perhaps most importantly, purchasing the disc set is an excellent way to directly support the Slackware Linux Project (and help us buy nachos).

Table 3-1. Slackware, Inc. contact information

Method	Information
telephone	1-800-786-9907
website	http://www.slackware.com
email	< orders@slackware.com >
snail mail	4041 Pike Lane, Suite F Concord, CA 94520-1207

Via the Internet

Slackware Linux is also freely available over the Internet. You may email in your support questions, but higher priority will be given to those who have purchased the official CD set.

The official Slackware Linux Project website is located at:

<http://www.slackware.com/>

The primary FTP location for Slackware Linux is:

`ftp://ftp.slackware.com/pub/slackware/`

[Prev](#)

Installation

[Home](#)

[Up](#)

[Next](#)

System Requirements

System Requirements

An easy Slackware installation requires, at minimum, the following:

Table 3-2. System Requirements

Hardware	Requirement
Processor	386
RAM	16 MB
Disk Space	500MB
Floppy Drive	1.44 MB

If you have the bootable CD, you will probably not need a floppy drive. Of course, it stands to reason that if you plan to install from CD you will need a CD-ROM drive. A network card is required for an NFS install. See [the section called NFS](#) for more information.

The disk space requirement is somewhat tricky. The 500MB recommendation is usually safe, but if you do a full install, you will need around one gigabyte of available hard disk space. Most users don't do a full install. In fact, many run Slackware on as little as 100MB of hard disk space.

Slackware can be installed to systems with less RAM and smaller hard drives, but doing so will require a little elbow grease. If you're up for a little work, take a look at the `LOWMEM.TXT` file in the distribution tree for a few helpful hints.

The Software Series

For reasons of simplicity, Slackware has historically been divided into software series. Once called disk sets because they were designed for floppy-based installation, the software series are now used primarily to categorize the packages included in Slackware. Today, floppy installation is still possible for the A and most of the N series (see below).

The following is a brief description of each software series.

Table 3-3. Software Series

Series	Contents
A	The base system. Contains enough software to get up and running and have a text editor and basic communication program.
AP	Various applications that do not require the X Window System.
D	Program development tools. Compilers, debuggers, interpreters, and man pages are all here.
DES	Includes the GNU libc <code>crypt()</code> function.
E	GNU emacs.
F	FAQs, HOWTOs, and other miscellaneous documentation.
GTK	The GNOME desktop environment, GTK widget library, and the GIMP.
K	The source code for the Linux kernel.
KDE	The K Desktop Environment. An X environment which shares a lot of look-and-feel features with the MacOS and Windows. The Qt library, which KDE requires, is also in this series.
N	Networking programs. Daemons, mail programs, telnet, news readers, and so on.
T	teTeX document formatting system.
TCL	The Tool Command Language. Tk, TclX, and TkDesk.
X	The base X Window System.

XAP	X Applications that are not part of a major desktop environment (for example, Ghostscript and Netscape).
XD	X11 program development. Libraries, server link kit, and PEX support.
XV	XView libraries, the OpenLook Virtual and Non-Virtual Window Managers, and various other XView applications.
Y	Games

Installation Methods

Floppy

While it was once possible to install all of Slackware Linux from floppy disks, the increasing size of software packages (indeed, of some individual programs) has forced the abandonment of the floppy install for all but two of the software series. The A series is still fully installable from floppy disks, and most of the N series is as well. This will give you a very basic system which can be used to install the rest of the distribution via a network.

Please note that floppy disks are still required for a CD-ROM install if you do not have a bootable CD, as well as for an NFS install.

CD-ROM

If you have the bootable CD, available in the official disc set published by Slackware, Inc. (see [the section called *Getting Slackware*](#)), a CD-based installation will be a bit simpler for you. If not, you will need to boot from floppies. Also, if you have special hardware that makes usage of the kernel on the bootable CD problematic, you may need to use specialized floppies.

See [the section called *Boot Disk*](#) through [the section called *Supplemental Disk*](#) for information on choosing and creating floppies from which to boot, if necessary.

NFS

NFS (the Network File System) is a way of making filesystems available to remote machines. An NFS install allows you to install Slackware from another computer on your network. The machine from which you are installing needs to be configured to export the Slackware distribution tree to the machine to which you're installing. This, of course, involves some knowledge of NFS, which is covered in [the section called *NFS \(Network File System\)*](#) in Chapter 5.

It is possible to perform an NFS install via such methods as PLIP (over a parallel port), SLIP, and PPP (though not over a modem connection). However, we recommend the use of a network card if available. After all, installing an operating system through your printer port is going to be a very, very slow process.

Boot Disk

The boot disk is the floppy you actually boot from to begin the installation. It contains a compressed kernel image which is used to control the hardware during installation. Therefore, it is very much required (unless you're booting from CD, as is discussed in [the section called *CD-ROM*](#)). The boot disks are located in the `bootdsk.s.144/` directory in the distribution tree.

There are more Slackware boot disks than you can shake a stick at (which is to say about 60). A complete list of boot disks, with a description of each, is available in the Slackware distribution tree in the file `bootdsk.s.144/WHICH.ONE`. However, most people are able to use the bare `.i` (for IDE devices) or `scsi.s` (for SCSI devices) boot disk image.

See [the section called *Making the Disks*](#) for instructions on making a disk from an image.

After booting, you will be prompted to insert the root disk. We recommend that you just humor the boot disk and

play along.

Root Disk

The root disk contains the setup program and a filesystem which is used during installation. It is also required. The root disk images are located in the directory `rootdsk`s in the distribution tree.

Fortunately, there are considerably fewer root disk images than there are boot disks. In fact, there are only three.

- `color.gz` is the one most people use. It's in color, which is nice.
- `text.gz` is just like `color.gz`, only it's not in color. Go figure.
- `umsdos.gz` is used for installing to a FAT (Windows) partition, which is generally recommended only for experimental purposes. For those interested in trying Slackware on a Windows partition, we recommend using ZipSlack or BigSlack.

Supplemental Disk

A supplemental disk is needed if you are performing an NFS install or installing to a system with PCMCIA devices. Supplemental disks are in the `rootdsk`s directory in the distribution tree, with the filenames `network.dsk` and `pcmcia.dsk`.

The root disk will instruct you on the use of supplemental disks when it is loaded.

Making the Disks

Once you've selected a boot disk image, you need to put it on a floppy. The process is slightly different depending on which operating system you're using to make the disks. If you're running Linux (or pretty much any Unix-like OS) you'll need to use the **dd**(1) command. Assuming `hejaz.dsk` is your disk image file and your floppy drive is `/dev/fd0`, the command to make a `hejaz.dsk` floppy is:

```
# dd if=hejaz.dsk of=/dev/fd0
```

If you're running a Microsoft OS, you'll need to use the **RAWRITE.EXE** program, which is included in the distribution tree in the same directories as the floppy images. Again assuming that `hejaz.dsk` is your disk image file and your floppy drive is A:, open a DOS prompt and type the following:

```
c:\ rawrite a: hejaz.dsk
```

Partitioning

After booting from your preferred media, you will need to partition your hard disk. The disk partition is where the Linux filesystem will be created and is where Slackware will be installed. At the very minimum we recommend creating two partitions; one for your root filesystem (`/`) and one for swap space.

After the root disk finishes loading, it will present you with a login prompt. Log in as **root** (there is no password). At the shell prompt, run either **cfdisk**(8) or **fdisk**(8). The **cfdisk** program provides a more user-friendly interface than the regular **fdisk** program, but does lack some features. We will briefly explain the **fdisk** program below.

Begin by running **fdisk** for your hard disk. In Linux, the hard disks do not have drive letters, but are represented by a file. The first IDE hard disk (primary master) is `/dev/hda`, the primary slave is `/dev/hdb`, and so on. SCSI disks follow the same type system, but are in the form of `/dev/sdX`. You will need to start **fdisk** and pass it your hard disk:

```
# fdisk /dev/hda
```

Like all good Unix programs, **fdisk** gives you a prompt (thought you were getting a menu, right?). The first

thing you should do is examine your current partitions. We do that by typing **p** at the **fdisk** prompt:

```
Command (m for help): p
```

This will display all sorts of information about your current partitions. Most people pick a free drive to install to and then remove any existing partitions on it to create room for the Linux partitions.



IT IS VERY IMPORTANT THAT YOU BACK UP ANY INFORMATION YOU WANT TO SAVE BEFORE DESTROYING THE PARTITION IT LIVES ON.

There is no easy way to recover from deleting a partition, so always back up before playing with them.

Looking at the table of partition information you should see a partition number, the size of the partition, and its type. There's more information, but don't worry about that for now. We are going to delete all of the partitions on this drive to create the Linux ones. We run the **d** command to delete those:

```
Command (m for help): d
Partition number (1-4): 1
```

This process should be continued for each of the partitions. After deleting the partitions we are ready to create the Linux ones. We have decided to create one partition for our root filesystem and one for swap. It is worth noting that Unix partitioning schemes are the subject of many flame wars, and that most users will tell you the *best* way to do it. Our advice is to make two partitions to start with, one for the root filesystem and one for swap space. Over time you will learn a partitioning scheme that suits your system.

Now we create the partitions with the **n** command:

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (0-1060, default 0): 0
Last cylinder or +size or +sizeM or +sizeK (0-1060, default 1060): +64M
```

You need to make sure you create primary partitions. The first partition is going to be our swap partition. We tell **fdisk** to make partition number 1 a primary partition. We start it at cylinder 0 and for the ending cylinder we type **+64M**. This will give us a 64 megabyte partition for swap. (The size of the swap partition you need actually depends on the amount of RAM you have. It is conventional wisdom that a swap space double the size of your RAM should be created.) Then we define primary partition number 2 starting at the first available cylinder and going all the way to the end of the drive.

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (124-1060, default 124): 124
Last cylinder or +size or +sizeM or +sizeK (124-1060, default 1060): 1060
```

We are almost done. We need to change the type of the first partition to type 82 (Linux swap). Type **t** to change the type, select the first partition, and type **82**. Before writing your changes to the disk, you should look at the new partition table one last time. Use the **p** in **fdisk** to display the partition table. If everything looks good, type **w** to write your changes to the disk and quit **fdisk**.

The setup Program

Once you have created your partitions, you are ready to install Slackware. The next step in the installation process is running the **setup**(8) program. To do so, simply type **setup** at the shell prompt. **setup** is a menu-driven system for actually installing the Slackware packages and configuring your system.

```
----- Slackware Linux Setup (version 7.1.0) -----
Welcome to Slackware Linux Setup.
Select an option below using the UP/DOWN keys and SPACE or ENTER.
Alternate keys may also be used: '+', '-', and TAB.

HELP      Read the Slackware Setup HELP file
KEYMAP    Remap your keyboard if you're not using a US one
ADDSWAP   Set up your swap partition(s)
TARGET    Set up your target partitions
SOURCE    Select source media
SELECT    Select categories of software to install
INSTALL   Install selected software
CONFIGURE Reconfigure your Linux system
EXIT      Exit Slackware Linux Setup

< OK >      <Cancel>
```

The setup process goes something like this: You step through each option in the **setup** program, in the order they are listed. (Of course, you are free to do things in almost any order you choose, but chances are it isn't going to work out very well.) Menu items are selected using the up and down arrow keys, and the **Okay** and **Cancel** buttons can be chosen by using the left and right arrow keys. Alternately, each option has a corresponding key, which is highlighted in the option name. Options which are flaggable (those indicated with a [X]) are toggled using the spacebar.

Of course, all of that is described in the **help** section of **setup**, but we believe in giving our readers their money's worth.

HELP

If this is your first time installing Slackware, you might want to take a look at the help screen. It will give a description of each part of **setup** (much like the one we're writing now, but less involved) and instructions for navigating the rest of the install.

```
----- Slackware Setup Help -----
Slackware Linux Help
-----

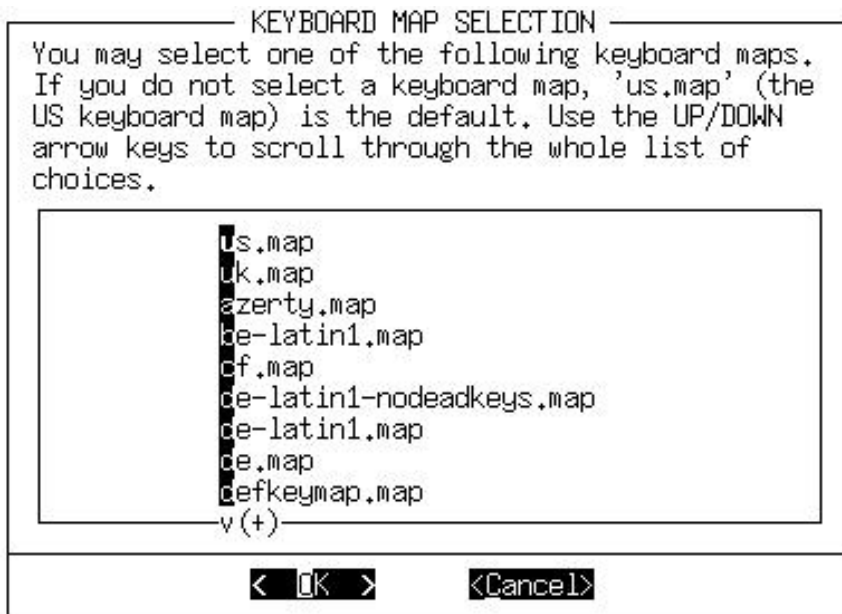
First, a little help on help. Whenever you encounter a text viewer
like this during the installation, you can move around with these
commands:

PGDN/SPACE    - Move down one page
PGUP/'b'      - Move up one page
ENTER/DOWN/'j' - Move down one line
UP/'k'        - Move up one line
LEFT/'h'      - Scroll left
RIGHT/'l'     - Scroll right
'o'           - Move to beginning of line
HOME/'g'      - Move to beginning of file
END/'G'       - Move to end of file
'/'          - Forward search
'?           - Backward search

----- ( 19%) -----
< EXIT >
```

KEYMAP

If you require a keymap other than the United States `qwerty` layout, you may want to take a look at this section. It offers a number of alternate layouts for your keyboarding enjoyment.



ADDSWAP

If you created a swap partition (back in [the section called *Partitioning*](#)), this section will allow you to enable it. It will autodetect and display the swap partitions on your hard drive, allowing you to select one to format and enable.

TARGET

The target section is where your other (non-swap) partitions are formatted and mapped to filesystem mount points. A list of the partitions on your hard disk will be displayed. For each partition, you will be given the option of whether to format (and if so, whether to check for bad blocks) and a selection of inode sizes to choose from. For normal use, the default inode size is fine.

The first option in the target section is the selection of a partition on which to install your root (`/`) filesystem. After that, you will be able to map other partitions to filesystems as you choose. (For instance, you may want your third partition, say `/dev/hda3`, to be your home filesystem. This is just an example; map the partitions as you see fit.)

SOURCE

The source section is where you select the source media from which you are installing Slackware. Currently there are four sources to choose from. These are floppy, CD-ROM, NFS, or a premounted directory.

SOURCE MEDIA SELECTION

Where do you plan to install Slackware Linux from?

1	Install from a hard drive partition
2	Install from floppy disks
3	Install via NFS
4	Install from a pre-mounted directory
5	Install from CD-ROM

< OK > <Cancel>

The floppy selection starts the prompting of many many diskettes. This option requires a lot of time and patience, but it is possible. Keep in mind that you need to make the floppies before you begin the setup program.

The CD-ROM selection enables a CD-ROM based installation. It will offer the option of scanning for a CD-ROM drive or displaying a list from which you can pick your type drive. Make sure you have the Slackware CD in your drive before allowing it to scan. After the program finds your CD-ROM drive it will ask if you want to perform a slakware or slaktest install. The default is slakware, which is a standard installation. The slaktest option installs a minimal set of software to the hard disk and keeps the majority of it on the CD. You will need the live CD in the official CD set for this option to work.

The NFS selection prompts for your network information and the network information for your NFS server. The NFS server must be set up in advance. Also note that you cannot use hostnames, you must use the IP addresses for both your machine and the NFS server (there is no name resolver on the setup disk).

The premounted directory offers the most flexibility. You can use this method to install from things such as Jaz disks, NFS mounts over PLIP, and FAT filesystems. Mount the filesystem to a location of your choosing before running setup, then specify that location here.

SELECT

The select option allows you to select the software series that you wish to install. These series are described in [the section called *The Software Series*](#). Please note that you must install the A series to have a working base system. All other series are optional.

PACKAGE SERIES SELECTION

Now it's time to select which general categories of software to install on your system. Use the spacebar to select or unselect the software you wish to install. You can use the up and down arrows to see all the possible choices. Recommended choices have been preselected. Press the ENTER key when you are finished.

<input checked="" type="checkbox"/>	A	Base Linux system
<input checked="" type="checkbox"/>	AP	Various Applications that do not need X
<input checked="" type="checkbox"/>	D	Program Development (C, C++, Lisp, Perl, etc.)
<input checked="" type="checkbox"/>	DES	GNU libc crypt() add-on
<input checked="" type="checkbox"/>	E	GNU Emacs
<input checked="" type="checkbox"/>	F	FAQ lists, HOWTO documentation
<input checked="" type="checkbox"/>	GTK	GTK+ and GNOME programs for X
<input checked="" type="checkbox"/>	K	Linux kernel source
<input checked="" type="checkbox"/>	KDE	Qt and the K Desktop Environment for X

v(+)

< OK > <Cancel>

INSTALL

Assuming that you have gone through the `target` , `source` , and `select` options, the `install` option will allow you to select packages from your chosen software series. If not, it will prompt you to go back and complete the other sections of the setup program. This option allows you to select from six different installation methods: *full*, *newbie*, *menu*, *expert*, *custom*, and *tag path*.

SELECT PROMPTING MODE

Now you must select the type of prompts you'd like to see during the installation process. If you have the drive space, the 'full' option is quick, easy, and by far the most foolproof choice. The 'newbie' mode provides the most information but is much more time-consuming (presenting the packages one by one) than the menu-based choices. Otherwise, you can pick packages from menus using 'expert' or 'menu' mode. Which type of prompting would you like to use?

full	Install everything (up to 386 MB of software)
newbie	Use verbose prompting (and follow tagfiles)
menu	Choose groups of packages from interactive menus
expert	Choose individual packages from interactive menus
custom	Use custom tagfiles in the package directories
tagpath	Use tagfiles in the subdirectories of a custom path
help	Read the prompt mode help file

< OK >

<Cancel>

The *full* option will install every package from all the software series that you chose in the `select` section. There is no further prompting. This is the easiest installation method, since you do not need to make any decisions on the actual packages to install. Of course, this option also takes up the most hard drive space.

The next option is *newbie*. This option installs all of the required packages in the selected series. For all other packages, it offers a prompt where you can select `Yes` , `No` , or `Skip` . `Yes` and `No` do the obvious, while `Skip` will go ahead to the next software series. Additionally, you will see a description and size requirement for each package to help you decide if you need it. We recommend this option for new users, as it ensures that you get all the required packages installed. However, it is a little slow because of the prompting.

Menu is a faster and more advanced version of the *newbie* option. For each series, a menu is displayed, from which you can select all the non-required packages you want to install. Required packages are not displayed on this menu.

For the more advanced user, `install` offers the *expert* option. This allows you complete control over what packages get installed. You can deselect packages that are absolutely required, resulting in a broken system. On the other hand, you can control exactly what goes onto your system. Simply select the packages from each series that you want installed. This is not recommended for the new user, as it is quite easy to shoot yourself in the foot.

The *custom* and *tag path* options are also for advanced users. These options allow you to install based upon custom tag files that you created in the distribution tree. This is useful for installing to large numbers of machines fairly quickly. For more information on using tag files, see [the section called Making Tags and Tagfiles \(for setup\) in Chapter 16](#).

After selecting your installation method, one of a few things will happen. If you selected `full` or `menu`, a menu screen will appear, allowing you to select the packages to be installed. If you selected `full`, packages will immediately start getting installed to the target. If you selected *newbie*, packages will be installed until an optional package is reached.

Note that it is possible to run out of space while installing. If you selected too many packages for the amount of free space on the target device, you will have problems. The safest thing to do is to select some software and add more later, if you need it. This can easily be done using Slackware's package management tools. For this

information, see [Chapter 16](#).

CONFIGURE

The configure section allows you to do some basic system configuration, now that the packages have been installed. What you see here depends in large part upon which software you have installed. You will, however, always see the following:

Kernel selection

Here you will be asked to select a kernel to install. You can install the kernel from the boot disk you used to install, the Slackware CD-ROM, or from another floppy which you (always thinking ahead) have prepared. Or you can elect to skip, in which case the default kernel will be installed and play will continue to the dealer's left.

INSTALL LINUX KERNEL

In order for your system to boot correctly, a kernel must be installed. If you've made it this far using the installation bootdisk's kernel, you should probably install it as your system kernel (/vmlinuz). If you're sure you know what you're doing, you can also install your choice of kernels from the Slackware CD, or a kernel from a floppy disk. You can also skip this menu, using whatever kernel has been installed already (such as a generic kernel from the A series.) Which option would you like?

bootdisk	Use the kernel from the installation bootdisk
cdrom	Use a kernel from the Slackware CD
floppy	Install a zimage or bzimage file from a DOS floppy
skip	Skip this menu and use the default /vmlinuz

< OK >

<Cancel>

Make a boot disk

Making a boot disk for future use is probably a good idea. You will have the option of formatting a floppy and then creating one of two types of boot disk. The first type, *simple*, simply (go figure) writes a kernel to the floppy. A more flexible (and highly recommended) option is *lilo*, which will of course create a lilo boot disk. See [the section called LILO in Chapter 7](#) for information on lilo. Of course, you may also choose to simply **continue**, in which case no boot disk will be made.

MAKE BOOTDISK

It is highly recommended that you make a bootdisk (or two) for your system at this time. There are two types of bootdisks that you can make: a simple bootdisk (which is just a kernel image written directly to disk) or a LILO bootdisk (which is more flexible, but takes a little longer to load). Which option would you like?

format	format floppy disk in /dev/fd0
simple	make simple vmlinuz > /dev/fd0 bootdisk
lilo	make lilo bootdisk
continue	leave bootdisk menu and continue with the configuration

< OK >

<Cancel>

Modem

You will be prompted for modem information. More specifically, you will be asked whether you have a modem, and if so, what serial port it is on.

MODEM CONFIGURATION

This part of the configuration process will create a /dev/modem link pointing to the serial device (ttyS0, ttyS1, ttyS2, ttyS3) representing your default modem. You can change this link later if you move your modem to a different port. Please select the serial device which you would like to use for your modem:

/dev/ttyS0	(COM1: under DOS)
/dev/ttyS1	(COM2: under DOS)
/dev/ttyS2	(COM3: under DOS)
/dev/ttyS3	(COM4: under DOS)
no modem	I don't have a modem!

These next configuration subsections may or may not appear, depending on whether or not you installed their corresponding packages. (**setup** is all kinds of adaptive, yo.)

Timezone

This one's pretty straightforward: you will be asked what time zone you are in. If you operate on Zulu time, we are very sorry; the (extremely long) list is alphabetically ordered, and you're at the bottom.

TIMEZONE CONFIGURATION

Please select one of the following timezones for your machine:

US/Alaska
US/Aleutian
US/Arizona
US/Central
US/East-Indiana
US/Eastern
US/Hawaii
US/Indiana-Starke
US/Michigan
US/Mountain
US/Pacific
US/Samoa
Africa/Abidjan
v(+)

Mouse

This subsection simply asks what kind of mouse you have, and whether you want **gpm**(8) (console mouse support) enabled on bootup.

— MOUSE CONFIGURATION —

This part of the configuration process will create a /dev/mouse link pointing to your default mouse device. You can change the /dev/mouse link later if the mouse doesn't work, or if you switch to a different type of pointing device. We will also use the information about the mouse to set the correct protocol for gpm, the Linux mouse server. Please select a mouse type from the list below:

bare	2 button Microsoft compatible serial mouse
ms	3 button Microsoft compatible serial mouse
mman	Logitech serial MouseMan and similar devices
ps2	PS/2 port mouse (also most laptops)
msc	MouseSystems serial (most 3 button mice)
pnp	Plug and Play (serial mice that do not work with ms)
ms3	Microsoft serial Intellimouse
imps2	Microsoft PS/2 Intellimouse

v(+)

< OK > <Cancel>

Hardware clock

This subsection asks if your computer's hardware clock is set to Coordinated Universal Time (UTC or GMT). Most PCs are not, so you should probably say no.

— HARDWARE CLOCK SET TO UTC? —

Is the hardware clock set to Coordinated Universal Time (UTC/GMT)? If it is, select YES here. If the hardware clock is set to the current local time (this is how most PCs are set up), then say NO here. If you are not sure what this is, you should answer NO here.

NO	Hardware clock is set to local time
YES	Hardware clock is set to UTC

< OK > <Cancel>

Font

The font subsection allows you to choose from a list of custom console fonts.

— SELECT A SCREEN FONT —

Select one of the following custom fonts. If you decide you like it, you can make it your new default screen font. You'll be able to try as many of these as you like.

t.fnt.gz
Agafari-12.psfu.gz
Agafari-14.psfu.gz
Agafari-16.psfu.gz
Cyr_a8x14.gz
Cyr_a8x16.gz
Cyr_a8x8.gz
Goha-12.psfu.gz
Goha-14.psfu.gz
Goha-16.psfu.gz

v(+)

< OK > <Cancel>

Here you are prompted for installation of LILO (the LInux LOader; see [the section called LILO in Chapter 7](#) for more information). If Slackware is to be the only operating system on your computer, *simple* should work just fine for you. If you are dual-booting, the *expert* option is a better choice. See [the section called Dual Booting in Chapter 7](#) for more information on dual-booting. The third option, *do not install*, is not recommended unless you know what you're doing and have a very good reason for not installing LILO. If you are performing an expert install, you will be given a choice as to where LILO will be put. You may place LILO in the MBR (Master Boot Record) of your hard drive, in the superblock of your root Linux partition, or on a floppy disk.

Please note that if you are currently using another operating system's boot loader it is advisable to install LILO either to the superblock of your root Linux partition or to a floppy. Installing to the MBR in such a case will obliterate the other operating system's boot loader and can make life very difficult.

Network

The network configuration subsection is actually **netconfig**. See [the section called netconfig in Chapter 5](#) for more information.

CD-ROM

The CD-ROM subsection simply asks if you would like the system to automatically scan for and mount an available CD-ROM disc in /cdrom.

CD-ROM device /dev/scd0 detected

A symbolic link has been created in the /dev directory setting /dev/scd0 as your default CD-ROM drive. Would you like to have your Linux startup scripts check this drive for a CD-ROM at boot time, and mount the disc on /cdrom if found? (NOTE: if you use this option, your system will complain if you boot without a CD, which may prove inconvenient, especially on systems that can boot a CD where you may not want the disc inserted at boot)

NO

No thanks, I'll mount the CD if/when I need it

YES

Yes, check for and mount the CD at boot time

< OK >

<Cancel>

X Window Manager

This subsection will allow you to choose a default window manager for X. See [Chapter 6](#) for more details on X and window managers.

SELECT DEFAULT WINDOW MANAGER FOR X

Please select the default window manager to use with the X Window System. This will define the style of graphical user interface the computer uses. KDE and GNOME provide the most features. People with Windows or MacOS experience will find KDE easy to use. GNOME is not quite as easy to use, but is very highly configurable and graphically rich. Other window managers are easier on system resources, or provide other unique features.

(X)

xinitrc.kde

KDE: K Desktop Environment

()

xinitrc.gnome

GNOME: GNU Network Object Model Environment

()

xinitrc.e

Enlightenment

v(+)

< OK >

<Cancel>

No matter which packages you installed, the last thing configure will do is ask you whether you want to go ahead and set a root password. For security reasons, this is probably a good idea; however, like almost everything else in Slackware, this is your call.

EXIT

The very existence of this section in the book is an insult to your intelligence. We humbly apologize and beg your forgiveness.

[Prev](#)

Installation

[Home](#)

[Up](#)

[Next](#)

Summary

Summary

You should now have Slackware Linux installed on your system. In addition, you should have some familiarity with partitioning devices, software packages, the **setup** program, and some simple configuration options. With this knowledge, you should be ready to get to work finishing up the configuration of your system.

III. Configuration

Table of Contents

- 4. [System Configuration](#)
 - 5. [Network Configuration](#)
 - 6. [The X Window System](#)
 - 7. [Booting](#)
-

Chapter 4. System Configuration

Table of Contents

[System Overview](#)

[Selecting A Kernel](#)

[Summary](#)

Before you can configure the more advanced parts of your system, it's a good idea to learn how the system is organized and what commands can be used to search for files and programs. It's also good to know if you need to compile a custom kernel and what the steps for doing that are. This chapter will familiarize you with system organization and configuration files. Then, you can move on to configuring the more advanced parts of the system.

System Overview

It's important to understand how a Linux system is put together before diving into the various configuration aspects. A Linux system is significantly different from a DOS or Windows system (or even a Macintosh), but these sections will help you get acquainted with the layout so that you can easily configure your system to meet your needs.

File System Layout

The first noticeable difference between Slackware Linux and a DOS or Windows system is the filesystem. For starters, we do not use drive letters to denote different partitions. Under Linux, there is one main directory. You can relate this to the C: drive under DOS. Each partition on your system is mounted to a directory on the main directory. It's kind of like an ever-expanding hard disk.

We call the main directory the root directory, and it's denoted with a single slash (/). This concept may seem strange, but it actually makes life easy for you when you want to add more space. For example, let's say you run out of space on the drive that has /home on it. Most people install Slackware and make one big root drive. Well, since a partition can be mounted to any directory, you can simply go to the store and pick up a new hard drive and mount it to /home. You've now grafted on some more space to your system. And all without having to move many things around.

Below, you will find descriptions of the major top level directories under Slackware.

/bin

Essential user programs are stored here. These represent the bare minimum set of programs required for a user to use the system. Things like the shell and the filesystem commands (`ls`, `cp`, and so on) are stored here. The `/bin` directory usually doesn't receive modification after installation. If it does, it's usually in the form of package upgrades that we provide.

`/boot`

Files that are used by the Linux Loader (LILO). This directory also receives little modification after an installation.

`/cdrom`

Remember that all drives have to be mounted to a directory on the main root directory? Well, `/cdrom` is provided as a mount point for your CD-ROM drive.

`/dev`

Everything in Linux is treated as a file, even hardware devices like serial ports, hard disks, and scanners. In order to access these devices, a special file called a device node has to be present. All device nodes are stored in the `/dev` directory. You will find this to be true across many UNIX-like operating systems.

`/etc`

This directory holds system configuration files. Everything from the X Window configuration file, the user database, to the system startup scripts. The system administrator will become quite familiar with this directory over time.

`/home`

Linux is a multiuser operating system. Each user on the system is given an account and a unique directory for personal files. This directory is called the user's home directory. The `/home` directory is provided as the default location for user home directories.

`/lib`

System libraries that are required for basic operation are stored here. The C library, the dynamic loader, the ncurses library, and kernel modules are among the things stored here.

`/lost+found`

When the system boots, the filesystems are checked for any errors. If errors are detected, the **fsck** program is run to see if any can be corrected. The corrected parts of the filesystem are written to the `/lost+found` directory.

`/mnt`

This directory is provided as a temporary mount point for working on hard disks or removable drives.

`/opt`

Optional software packages. The idea behind `/opt` is that each software package installs to `/opt/<software package>`, which makes it easy to remove later.

Slackware distributes some things in `/opt` (such as KDE in `/opt/kde`), but you are free to add anything you want to `/opt`.

`/proc`

This is a unique directory. It's not really part of the filesystem, but a virtual filesystem that provides access to kernel information. Various pieces of information that the kernel wants you to know are conveyed to you through files in the `/proc` directory. You can also send information to the kernel through some of these files. Try doing `cat /proc/cpuinfo`.

`/root`

The system administrator is known as `root` on the system. `root`'s home directory is kept in `/root` instead of `/home/root`. The reason is simple. What if `/home` was a different partition from `/` and it could not be mounted? `root` would naturally want to log in and repair the problem. If his home directory was on the damaged filesystem, it would make it difficult for him to log in.

`/sbin`

Essential programs that are run by `root` and during the system bootup process are kept here. Normal users will not run programs in this directory.

`/tmp`

The temporary storage location. All users have read and write access to this directory.

`/usr`

This is the big directory on a Linux system. Everything else pretty much goes here, programs, documentation, the kernel source code, and the X Window system. This is the directory to which you will most likely be installing programs.

`/var`

System log files, cache data, and program lock files are stored here. This is the directory for frequently-changing data.

You should now have a good feel for which directories contain what on the filesystem. The next section will help you find specific files easily, so you don't have to do it by hand.

Finding Files

You now know what each directory holds, but it still doesn't really help you find things. I mean, you could go looking through directories, but there are quicker ways. There are four main file search commands available in Slackware.

which

The first is the **which**(1) command. **which** is usually used to locate a program quickly. It just searches your `PATH` and returns the first instance it finds and the directory path to it. Take this example:

```
$ which bash
/bin/bash
```

From that you see that `bash` is in the `/bin` directory. This is a very limited command for searching, since it only searches your `PATH`.

whereis

The **whereis**(1) command works similar to **which**, but can also search for man pages and source files. A **whereis** search for `bash` should return this:

```
$ whereis bash
bash: /bin/bash /usr/bin/bash /usr/man/man1/bash.1.gz
```

This command not only told us where the actual program, but also where the online documentation is stored. Still, this command is limited. What if you wanted to search for a specific configuration file? You can't use **which** or **whereis** for that.

find

The **find**(1) command will search for anything. I want to search the entire system for the default `xinitrc` file on the system.

```
$ find / -name xinitrc
./var/X11R6/lib/xinit/xinitrc
```

find will take a while to run, since it has to traverse the entire root directory tree. And if you run this command as a normal user, you will probably get permission denied error messages for directories that only root can see. But **find** found our file, so that's good. If only it could be a bit faster...

locate

The **locate**(1) command searches the entire filesystem, just like the `find` command can do, but it searches a database instead of the actual filesystem. The database is set to automatically update at 4:40AM, so you have a somewhat fresh listing of files on your system. You can manually run **updatedb**(1) to update the locate database (before running **updatedb** by hand, you must first **su** to the **nobody** user). Here's an example of **locate** in action:

```
$ locate xinitrc    # we don't have to go to the root
/var/X11R6/lib/xinit/xinitrc
/var/X11R6/lib/xinit/xinitrc.fvwm2
/var/X11R6/lib/xinit/xinitrc.openwin
/var/X11R6/lib/xinit/xinitrc.twm
```

We got more than what we were looking for, and quickly too. With these commands, you should be able to find whatever you're looking for on your Linux system.

The /etc/rc.d Directory

The system initialization files are stored in the `/etc/rc.d` directory. Slackware uses the BSD-style layout for its initialization files. Each task or runlevel is given its own rc file. This provides an organized structure that is easy to maintain.

There are several categories of initialization files. These are system startup, runlevels, network initialization, and System V compatibility. As per tradition, we'll lump everything else into an other category.

System Startup

The first program to run under Slackware besides the Linux kernel is **init**(8). This program reads the `/etc/inittab`(5) file to see how to run the system. It runs the `/etc/rc.d/rc.S` script to prepare the system before going into your desired runlevel. The `rc.S` file enables your virtual memory, mounts your filesystems, cleans up certain log directories, initializes Plug and Play devices, loads kernel modules, configures PCMCIA devices, sets up serial ports, and runs System V init scripts (if found). Obviously `rc.S` has a lot on its plate, but here are some scripts in `/etc/rc.d` that `rc.S` will call on to complete its work:

`rc.S`

This is the actual system initialization script.

`rc.modules`

Loads kernel modules. Things like your network card, PPP support, and other things are loaded here. If this script finds `rc.netdevice`, it will run that as well.

`rc.pcmcia`

Probes for and configures any PCMCIA devices that you might have on your system. This is most useful for laptop users, who probably have a PCMCIA modem or network card.

`rc.serial`

Configures your serial ports by running the appropriate **setserial** commands.

`rc.sysvinit`

Looks for System V init scripts for the desired runlevel and runs them. This is discussed in more detail below.

Runlevel Initialization Scripts

After system initialization is complete, **init** moves on to runlevel initialization. A runlevel describes the state that your machine will be running in. Sound redundant? Well, the runlevel tells **init** if you will be accepting multiuser logins or just a single user, whether or not you want network services, and if you will be using the X Window System or **agetty**(8) to handle logins. The files below define the different runlevels in Slackware Linux.

`rc.0`

Halt the system (runlevel 0). By default, this is symlinked to `rc.6`.

`rc.4`

Multiuser startup (runlevel 4), but in X11 with KDM, GDM, or XDM as the login manager.

`rc.6`

Reboot the system (runlevel 6).

`rc.K`

Startup in single user mode (runlevel 1).

`rc.M`

Multiuser mode (runlevels 2 and 3), but with the standard text-based login. This is the default runlevel in Slackware.

Network Initialization

Runlevels 2, 3, and 4 will start up the network services. The following files are responsible for the network initialization:

`rc.inet1`

Created by **netconfig**, this file is responsible for configuring the actual network interface.

`rc.inet2`

Runs after `rc.inet1` and starts up basic network services.

`rc.atalk`

Starts up AppleTalk services.

`rc.httpd`

Starts up the Apache web server.

`rc.samba`

Starts up Windows file and print sharing services.

`rc.news`

Starts up the news server.

System V Compatibility

System V init compatibility was introduced in Slackware 7.0. Many other Linux distributions make use of this style instead of the BSD style. Basically each runlevel is given a subdirectory for init scripts, whereas BSD style gives one init script to each runlevel.

The `rc.sysvinit` script will search for any System V init scripts you have in `/etc/rc.d` and run them, if the runlevel is appropriate. This is useful for certain commercial software packages that install System V init scripts and scripts for BSD style init.

Other Files

The scripts described below are the other system initialization scripts. They are typically run from one of the major scripts above, so all you need to do is edit the contents.

`rc.cdrom`

If enabled, this script will scan for a CD-ROM in a drive and mount it under `/cdrom` if it finds one.

`rc.gpm`

Starts up general purpose mouse services. Allows you to copy and paste at the Linux console.

`rc.ibcs2`

Starts up the Intel Binary Compatibility support. This is only needed if you plan on running programs compiled on SCO UNIX, or other commercial Intel UNIX flavors. It is not needed to run Linux programs.

`rc.font`

Loads the custom screen font for the console.

`rc.local`

Contains any specific startup commands for your system. This is empty after a fresh install, as it is reserved for local administrators. This script is run after all other initialization has taken place.

To enable a script, all you need to do is add the execute permissions to it with the **chmod** command. To disable a script, remove the execute permissions from it. For more information about **chmod**, see [the section called *Permissions* in Chapter 9](#).

[Prev](#)

Configuration

[Home](#)

[Up](#)

[Next](#)

Selecting A Kernel

Selecting A Kernel

The kernel is the part of the operating system that provides hardware access, process control, and overall system control. The kernel contains support for your hardware devices, so picking one for your system is an important setup step.

Slackware provides around sixty precompiled kernels that you can pick from, each with a standard set of drivers and additional specific drivers. You can run one of the precompiled kernels or you can build your own kernel from source. Either way, you need to make sure that your kernel has the hardware support your system needs.

The /kernels Directory On The Slackware CD-ROM

The precompiled Slackware kernels are available in the `/kernels` directory on the Slackware CD-ROM or on the FTP site in the main Slackware directory. The available kernels change as new releases are made, so the documentation in that directory is always the authoritative source. The `/kernels` directory has subdirectories for each kernel available. The subdirectories have the same name as their accompanying boot disk. In each subdirectory you will find the following files:

File	Purpose
<code>System.map</code>	The system map file for this kernel
<code>bzImage</code> (or <code>zImage</code>)	The actual kernel image
<code>config</code>	The source configuration file for this kernel

To use a kernel, copy the `System.map` and `config` files to your `/boot` directory and copy the kernel image to `/vmlinuz`. Run `/sbin/lilo(8)` to install LILO for the new kernel, and then reboot your system. That's all there is to installing a new kernel.

The kernels that end with a `.i` are IDE kernels. That is, they include no SCSI support in the base kernel. The kernels that end with `.s` are SCSI kernels. They include all the IDE support in `.i` kernels, plus SCSI support.

Compiling A Kernel From Source

The question "Should I compile a kernel for my system?" is often asked by new users. The answer is a definite maybe. There are few instances where you will need to compile a kernel specific to your system. Most users can use a precompiled kernel and the loadable kernel modules to achieve a fully working system. You will want to compile a kernel for your system if you are upgrading kernel versions to one that we do not currently offer in Slackware, or if you have patched the kernel source to get special device support that is not in the native kernel source.

Building your own kernel is not that hard. The first step is to make sure you have the kernel source installed on your system. Make sure that you installed the packages from the K series during the installation. You will also want to make sure you have the D series installed, specifically the C compiler, GNU make, and GNU binutils. In general, it's a good idea to have the entire D series installed if you plan on doing any kind of development. Now we are ready to build a kernel:

```
$ su -  
Password:  
# cd /usr/src/linux
```

The first step is to bring the kernel source into its base state. We issue this command to do that:

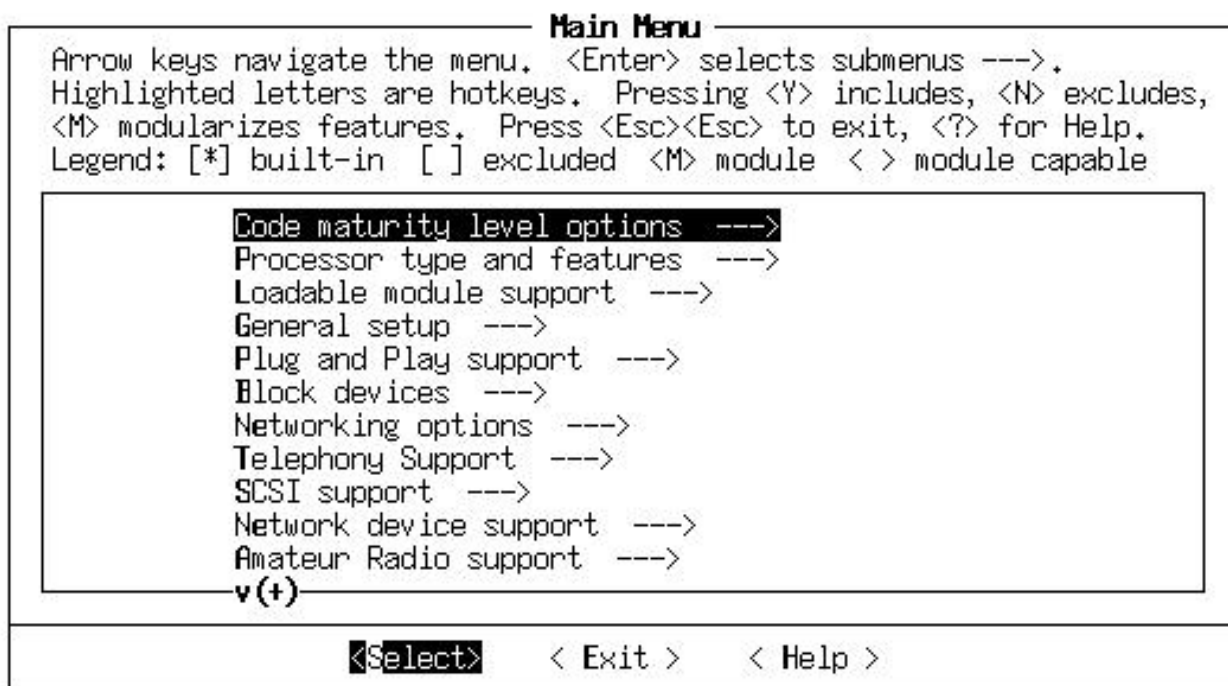
```
# make mrproper
```

Now you can configure the kernel for your system. The current kernel offers three ways of doing this. The first is the original text-based question and answer system. It asks a bunch of questions and then builds a configuration file. The problem with this method is that if you mess up, you must start over. The method that most people prefer is the menu driven one. Lastly, there is an X-based kernel configuration tool. Pick the one you want and issue the appropriate command:

```
# make config      (text-based Q&A version)  
# make menuconfig  (menu driven, text-based version)  
# make xconfig     (X-based version, make sure you are in X first)
```

Figure 4-1. The menu-driven kernel configuration program.

Linux Kernel v2.2.16 Configuration



New users will probably find **menuconfig** to be the easiest to use. Help screens are provided that explain the various parts of the kernel. After configuring your kernel, exit the configuration program. It will write the necessary configuration files. Now we can prepare the source tree for a build:

```
# make dep  
# make clean
```

The next step is to compile the kernel. First try issuing the **zImage** command below. That will fail if your kernel is too large. Don't worry, you can still build it with the **bzImage** command.

```
# make zImage      (try this first)  
# make bzImage     (if make zImage fails, issue this command)
```

This may take a while, depending on your CPU speed. During the build process, you will see the

compiler messages. After building the kernel image, you will want to build any parts of the kernel that you flagged as modular.

```
# make modules
```

We can now install the kernel and modules that you compiled. To install the kernel on a Slackware system, these commands should be issued:

```
# mv /vmlinuz /vmlinuz.old
# cat arch/i386/boot/zImage > /vmlinuz
# mv /boot/System.map /boot/System.map.old
# cp System.map /boot/System.map
```

Replace the **zImage** with **bzImage** if you had to build a big kernel. You will want to edit `/etc/lilo.conf` and add a section to boot your old kernel in case your new one does not work. After doing that, run `/sbin/lilo` to install the new boot block. You can now reboot with your new kernel.

Using Kernel Modules

Kernel modules are another name for device drivers that can be inserted into a running kernel. They allow you to extend the hardware supported by your kernel without needing to pick another kernel or compile one yourself.

Modules can also be loaded and unloaded at any time, even when the system is running. This makes upgrading specific drivers easy for system administrators. A new module can be compiled, the old one removed, and the new one loaded, all without rebooting the machine.

Modules are stored in the `/lib/modules/<kernel version>` directory on your system. They can be loaded at boot time through the `rc.modules` file. This file is very well commented and offers examples for major hardware components. To see a list of modules that are currently active, use the **lsmod(1)** command:

```
# lsmod
Module                Size  Used by
parport_pc            7220    0
parport               7844    0 [parport_pc]
```

You can see here that I only have the parallel port module loaded. To remove a module, you use the **rmmod(1)** command. Modules can be loaded by the **modprobe(1)** or **insmod(1)** command. **modprobe** is usually safer because it will load any modules that the one you're trying to load depends on.

A lot of users never have to load or unload modules by hand. They use the kernel autoloader for module management. All you have to do is uncomment the `/sbin/kerneld(8)` line in `/etc/rc.d/rc.modules` and the autoloader will start up. It will take care of loading and unloading modules as you request them. A request just involves trying to access that device.

More information can be found in the man pages for each of these commands, plus the `rc.modules` file.

Summary

You should now be familiar with commands for searching the filesystem, organization of the filesystem, and the configuration files in the `/etc` directory. These skills will be extremely helpful as you learn more about the system. In addition, you should know how to configure and compile a kernel from source.

Chapter 5. Network Configuration

Table of Contents

[Network Hardware](#)

[Network Utilities](#)

[The /etc files](#)

[rc.inet1](#)

[rc.inet2](#)

[NFS \(Network File System\)](#)

[tcp_wrappers](#)

[Summary](#)

Network Hardware

Like most interesting things you can do with a computer, attaching it to a network involves some specialized hardware. You'll need a NIC (Network Interface Card) to connect to a LAN, perhaps a modem to connect to an Internet provider, or perhaps both (or several of each, or none).

For the purposes of configuration, we can divide said hardware into PCMCIA (for laptops) and non-PCMCIA categories. The reason for this somewhat lopsided division is that currently PCMCIA hardware is not supported by the kernel distribution, but by a separate package which includes the necessary drivers (as kernel modules) and some software for configuration and management of PCMCIA devices. Everything else, of course, is handled by the standard kernel distribution.

netmods

The drivers for network devices that the kernel supports are included in the netmods package (`slackware/n3/netmods.tgz`). If you haven't installed netmods yet, you'll need to do so now. (See [Chapter 16](#) for help with installing packages.)

Kernel modules that are to be loaded on boot-up are loaded from the `rc.modules` file in `/etc/rc.d`. The default `rc.modules` file includes a Network device support section. If you open `rc.modules` and look for that section, you'll notice that it first checks for an executable `rc.netdevice` file in `/etc/rc.d`; `rc.netdevice` is created if **setup** successfully autoprobes for your network device during installation. If it did, you're

probably not reading this (ooh, paradox); if it didn't, read on.

Below that `if` block is a list of network devices and `modprobe` lines, each commented out. Find your device and uncomment the corresponding `modprobe` line, then save the file. Running `rc.modules` as root should now load your network device driver (as well as any other modules that are listed and uncommented). Note that some modules (such as the `ne2000` driver) require parameters; make sure you select the correct line.

PCMCIA network devices

PCMCIA network devices should be even easier than others. Make sure you have the `pcmcia` package (`slakware/all/pcmcia.tgz`) installed. (see [Chapter 16](#) for details on package installation.) Upon installation, the `pcmcia` package will create an `rc.pcmcia` file in `/etc/rc.d` and an `/etc/pcmcia` directory, and will install drivers to `/lib/modules/<kernel version>/pcmcia`. The cool thing about the `pcmcia` package is that it will attempt to autodetect the insertion and removal of supported pcmcia devices; you should be able to simply insert your pcmcia network adapter and listen for the beep it gives when loading the necessary modules. If you remove the card, its driver modules should be automatically removed.

Unfortunately, if you compile a newer kernel version you will probably have to recompile `pcmcia-cs` to get the drivers updated. Of course, the source is included; check the `source/a/pcmcia` directory for source, scripts, and any documentation we have to help you with that.

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

Network Utilities

Network Utilities

ifconfig

Now that your kernel can talk to your network hardware, what's needed is a way for software to tell the kernel to pass some information along, and vice versa. We need to configure an interface. We need **ifconfig**(8).

ifconfig is probably best learned by example; you may want to just take a look at your `rc.inet1` file (covered in [the section called `rc.inet1`](#)) to see how it's done there. The simplest and most common case looks something like this:

```
# ifconfig eth0 192.168.1.10 broadcast 192.168.1.255 \  
netmask 255.255.255.0
```

This line brings up `eth0` (the first ethernet interface; for token ring use `tr0`, ppp use `ppp0`, etc.) with an IP of 192.168.1.10, a broadcast address of 192.168.1.255 (the entire 192.168.1 subnet) and a netmask of 255.255.255.0 (indicating that all of the first three parts of the dotted quad of the IP address refer to the network, while all of the last part, .10, refers to the host). Unless you're doing something funky, you can almost always use a broadcast address that's the first three parts of your IP followed by 255. You can also almost always use a netmask of 255.255.255.0. If you *are* doing something funky, you probably know enough that this part of the book isn't going to be of much use to you.

ifconfig can also be used just to see what's there. Run it without any options or parameters to get a listing of all of your current network interfaces and their settings.

route

In order to know where to send which data, the kernel maintains a routing table. I'm not going to go into much detail about it here, but you can view the routing table by running `/sbin/route(8)`. **route -n** will give you the table with IP addresses instead of names; this is useful if you're having trouble talking to your name server or if you're just not interested in the illusory world of domain names. Fortunately, if you have a simple network setup (and most people do), the 2.2 kernels will automatically create the necessary routing table entries for you.

netconfig

netconfig is part of the Slackware setup program, but like most of **setup**'s components it can be run on its own. **netconfig** is very straightforward, and will walk you through setting up a basic network connection. It's an especially good program to use if you're not very familiar or comfortable with the network rc files. When you run **netconfig**, you'll be greeted with this screen:

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

< OK >

Next, you'll be prompted to enter the hostname and domain name of your computer. You can probably just make something up for both of these, unless you are setting up a server or other machine that lots of people will use. Then, you will be asked if you will be using a static IP, DHCP, or just loopback.

SETUP IP FOR 'nomex.tdn'

Now we need to know how your machine connects to the network. If you have an internal network card and an assigned IP address, gateway, and DNS, use the 'static IP' choice to enter these values. If your IP address is assigned by a DHCP server (commonly used by cable modem and DSL services), select 'DHCP'. If you do not have a network card, select the 'loopback' choice. 'loopback' is also the correct choice if your only connection to the network will be through a serial modem (with SLIP or PPP), or if you are using a laptop network card (these are configured in /etc/pcmcia/). What type of network connection best describes your machine?

s tatic IP	Use a static IP address to configure ethernet
D HCP	Use a DHCP server to configure ethernet
l oopback	Set up a loopback connection (modem or no net)

< OK >
<Cancel>

If you are not going to be connected to a network, choose loopback. If you are setting up a computer that will be attached to a university or large office network, you should most likely choose DHCP. Otherwise, choose static IP. Unless you chose static IP, you are now done. If you did choose static IP, you will now have to enter your computer's IP address, network mask, broadcast address, and name server address. **netconfig** will tell you how to figure out all of those numbers.

pppsetup

Slackware includes the **pppsetup** utility for configuring a dialup connection to an ISP. It is located in the `ppp.tgz` package in the N software series. **pppsetup** uses the same interface as the **setup** program. If you don't remember how to use this interface, refer back to [the section called The setup Program in Chapter 3](#) for some instructions. **pppsetup** asks you a series of questions and sets up several configuration files in `/etc/ppp` for you. As root, run **pppsetup**; we'll walk through the questions here.

PPPSETUP 1.98 on SLACKWARE.

Written by Robert S. Liesenfeld <xunil@bitstream.net> <IRC:Xunil>
Changes for 1.98 by Kent Robotti <robotti@erols.com>
Patched for Slackware by Patrick Volkerding <volkerdi@slackware.com>

You should get these docs if you don't already have them:

ftp://metalab.unc.edu/pub/Linux/docs/howto/PPP-HOWTO
ftp://metalab.unc.edu/pub/Linux/docs/faqs/PPP-FAQ

Press [Enter] to continue with pppsetup...

(100%)

< EXIT >

Phone number

The first question prompts for the phone number of your ISP, prefixed by the type of dialing. For most people, you will want to use tone dialing. If your ISP's phone number was 555-1013 and you used tone dialing, you would enter **atdt5551013** into the dialog box.

PHONE NUMBER ...

To begin setting up your PPP connection, i need to know a few things.
For starters, what is the phone number of your (I)nternet (S)ervice
(P)rovider?

Example: atdt6661776 <-For (t)one dialing.)
Example: atdp6661776 <-For (p)ulse dialing.)

Include the: atd? It's usally just: atdtphonenumber

(Note: in the USA, use atdt*70,6661776 [comma required!] to turn
off call waiting.)

< OK > <Cancel>

If you have call waiting on your phone line and want it disabled for when you connect (probably a good idea), make sure to enter something like this into the dialog box: **atdt*70,5551013**

The comma is required. It puts a 1.5 second pause between the *70 to disable call waiting and the ISP's phone number. It won't work without the comma.

Modem device

Next, select the location of your modem. If you know what COM device it was under Windows, you can select the listed equivalent. Otherwise, you might have to do some experimenting. The best course is to start at **ttys0** and work your way down the list.

MODEM DEVICE ...

Where is your modem /dev/ttyS?

ttyS0	= (COM1: under DOS)
ttyS1	= (COM2: under DOS)
ttyS2	= (COM3: under DOS)
ttyS3	= (COM4: under DOS)

< **OK** > <Cancel>

Modem baud rate

Next, pick the baud rate that's closest to your modem's. If you don't know the baud rate, you could check the modem's box or any documentation that came with it. Each selection has several examples, so it shouldn't be hard to figure out.

MODEM BAUD RATE ...

What baud rate is your modem?

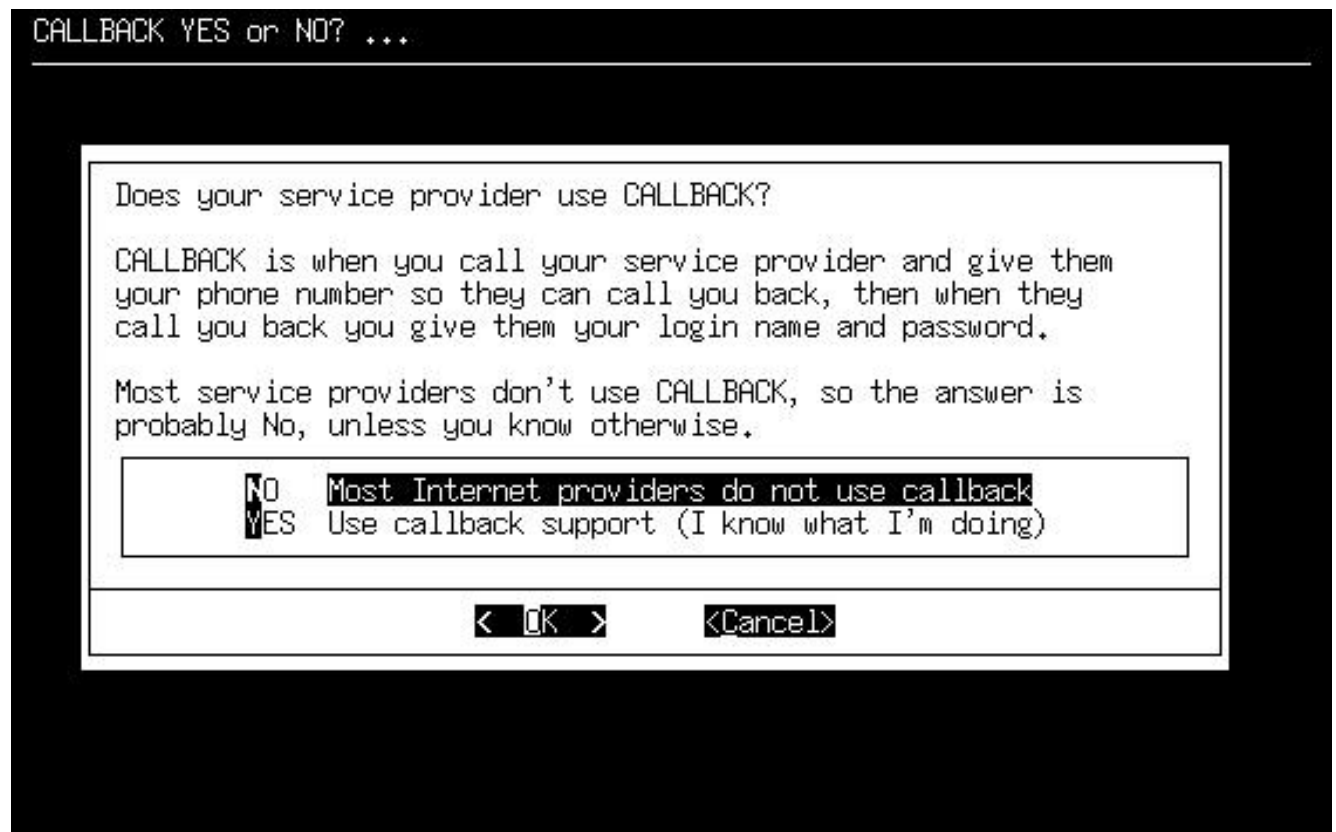
460800	460KBps	- ISDN modem...
230400	230KBps	- 56Kbps modem... or ISDN modem...
115200	115KBps	- 28.8, 33.6, or 56Kbps modem...
57600	57.6KBps	- 28.8, 33.6, or 56Kbps modem...
38400	38.4KBps	- Hangin ten on the net! 28.8 or 33.6...
19200	19.2KBps	- Better known as 14.4...
9600	9600bps	- No comment...

< **OK** > <Cancel>

Callback

Now, you'll need to start referring to any information that your ISP gave you. Few ISPs use callback, so you can probably safely select NO . Callback is where you dial up the ISP and they

then call you back so you can log in.



If you have to use callback, select YES . Then, you'll be prompted to enter your phone number, a login name, and a password. You may not have to enter an initial login name and password. Finally, you'll be asked what kind of authentication scheme your ISP uses. If they use CHAP or PAP, select YES . Later on, you'll need to set this up. Refer to the section below. If they don't use either of those, select NO and refer to the Chat script section below.

Modem init string

Unless you have a strange modem, you can probably just hit enter to select the default init string (AT&FH0). Otherwise, refer to any documentation that came with your modem as to what to use for the init string.

MODEM INIT STRING ...

The default modem init string will be: "AT&FH0" OK

If you want to change it, put your init string in the box below.

If you use \ in the init string, put it twice.
Example: "AT\&F\\K3\\N3H0" OK

M = No sound. S95=46 = Show CARRIER speed: 28800 etc.
Put "" around each init string with "&" in it.
Put OK after each init string. Example: ATZ OK "AT\&F1MH0" OK

Just press [Enter] on a empty box to accept the default above.

< OK >

<Cancel>

Domain name

Now, you'll need to enter the domain name of your ISP. This will be something of the form example.net , slackware.com , or something similar. (Okay, it's most definitely not going to be slackware.com :)

DOMAIN NAME ...

What is your (I)nternet (S)ervice (P)rovider's domain name?

This is usually something like...

Examples: something.edu something.net something.com something.org

< OK >

<Cancel>

DNS IP address

Your ISP should have provided you with the IP address of their nameserver. If you've got that IP address, punch it into the box. Otherwise, check in with your ISP to see what number to use.

DNS IP ADDRESS ...

What is the IP address of your Internet provider's nameserver?

It's important that these IP numbers be correct.
The IP numbers should not be: 0.0.0.0

Note: Your service provider's technical support can provide you
with this information. Example: 207.132.116.5

█

< OK >

<Cancel>

Authentication method

This question might also take some trial and error work. You need to figure out if your ISP uses CHAP, PAP, or neither for user authentication. The easiest way to figure this out is to call your ISP. However, if you are presented with login and password prompts when you connect, you should most likely choose "SCRIPT". Otherwise, check with your ISP to determine which one to use.

PAP, CHAP, or SCRIPT? ...

Does your service provider use PAP or CHAP?

If you're presented with a Username: or Login: and Password:
prompt when you connect to your service provider, they're
'probably' not using PAP or CHAP, so you can answer SCRIPT.
I said 'probably', the only way to know for sure is to ask your
service provider, this could save you a lot of wasted time.

PAP	AUTHENTICATION
CHAP	AUTHENTICATION
MS-CHAP-80	is microsoft's version of CHAP.
SCRIPT	Create Chat Script For Login.

< OK >

<Cancel>

PAP or CHAP

If you selected PAP or CHAP on the authentication method screen, you'll be prompted to enter

your username. Your ISP should have assigned you a username. If they didn't, there is something terribly wrong. You will need to contact them and get a username.

Then, you should enter the password that your ISP assigned to you in the next dialog box.

Chat script

If you selected **SCRIPT** on the authentication method screen, you'll be presented with a fairly lengthy discussion of what a chat script is. Make sure to read it because it describes everything very well. Basically, you want to tell it what kind of information your ISP is going to send and what your computer should send back to log in.

You'll then go into a loop of entering some text for what your computer should expect from your ISP, followed by what information it should send back. You can break this loop by hitting enter on an empty box.

Done

Finally, you'll be shown your complete ppp configuration files. You can't really do anything about them, but you can at least check everything out. Hit enter to save everything and leave **pppsetup**.

This screen also has a lot of information about how to bring up your dialup connection and how to drop it after you are done. The basic idea is this: as root, run **ppp-go** to start the connection. Once it gives you a local and remote IP, you are connected and on the internet. When you are done, run **ppp-off** as root and it will drop the connection.

```
===== DONE =====
=====
These are your PPP configuration files and instructions...
=====

# This is your /etc/ppp/pppscript.
Look at /etc/ppp/pppscript.

# This is your /etc/ppp/options file.

# General configuration options for PPPD:
lock
defaultroute
noipdefault
modem
/dev/ttyS2
57600
crtsets

( 9%)
< EXIT >
```

The /etc files

/etc/inetd.conf

On a network-centric operating system, it's not uncommon for a lot of services to be running. Typically for each service that is available a program needs to be sitting around listening for connections, which can get a bit burdensome on a system that runs a lot of these servers. In order to reduce overhead, `inetd` was created. `inetd` is the internet super-server -- it sits around and listens for connections on many sockets, and when one comes in `inetd` fires up the appropriate server to handle it. Thus the many waiting servers are reduced to one.

`/etc/inetd.conf` is the configuration file for `inetd`. It specifies which servers get run for which connections. The **inetd**(8) man page has more detailed information, of course, but let's take a quick look at a basic service line:

```
ftp stream tcp nowait root /usr/sbin/tcpd wu.ftpd -l -i -a
```

This is the line for the ftp server. Note the protocol name (`ftp`) first, and last the command to run to respond. In this case, the program that is run in response to a connection attempt is `/usr/sbin/tcpd`; it is a wrapper program that provides some basic security options for the server it wraps. **wu.ftpd** is our actual ftp server, but `tcpd` runs that for us. More information on `tcpd` is given in [the section called tcp wrappers](#).

Like many system files, lines in `inetd.conf` are commented by the `#` character; you can add and remove services from `inetd` simply by commenting their lines in or out and restarting **inetd**.

/etc/resolv.conf

This is the file that tells the rest of the system where to get its DNS information. Any name servers you use are listed here, as well as your host's domain name. Here's a sample `resolv.conf` (from the laptop I'm typing this on, `ninja.tdn`):

```
domain tdn
nameserver 192.168.1.1
search tdn. slackware.com
```

The first line describes `ninja`'s domain name; this is everything after the hostname in my address. The second is the DNS server for our house network. You can have as many of these as you need; they will be checked in order from first to last whenever a program needs to look up a domain name's corresponding IP address.

The last line is a bit more interesting. It describes any domain names that should be *assumed* by my system. For instance, suppose I have the machines `zuul.tdn` and `hejaz.slackware.com`. I can simply do **ping zuul** and **ping hejaz** to ping them, respectively. This is because ping first tries to add `.tdn` to `zuul`'s name, and finds a match and goes with it. In the case of `hejaz`, it first tries `hejaz.tdn`. There's no match, so it then tries `hejaz.slackware.com` -- bingo. Note that all domains listed in the **search** line need to end with a `'.'` *except* the last one; if there's only one, it is the last one and needs

no trailing '.

/etc/hosts

The `hosts` file allows the simplest kind of domain lookup. It is a list of hostnames and their corresponding IPs. This is useful in a small network where DNS is not worthwhile, in instances where DNS is unreliable, etc. and is used by the machine during boot time when no name servers are accessible. Mine might look like this:

```
127.0.0.1    localhost
192.168.1.32  ninja.tdn  ninja
```

The first line should be self-explanatory. The second, however, may not. You can list as many names and aliases for an address as you like, separated by a space. So I have `192.168.1.32` translated to `ninja.tdn` (and vice versa), but the alias `ninja` can also be used when I'm too lazy to type `.tdn` (which is most of the time).

[Prev](#)

Network Utilities

[Home](#)

[Up](#)

[Next](#)

rc.inet1

rc.inet1

`/etc/rc.d/rc.inet1` is the file used to set up network infrastructure -- it initializes devices and sets addresses and routes. The `rc.inet1` that ships with Slackware is pretty heavily commented, so we'd just be repeating ourselves if we went through it all again here.

rc.inet2

The `/etc/rc.d/rc.inet2` file is there for the other part of networking: setting up services and daemons and handling any interesting networking options. Let's look at a sample block:

```
# Start the NAMED/BIND name server:
if [ -f ${NET}/named ]; then
    echo -n " named"
    ${NET}/named -u daemon -g daemon
fi
```

The important line here is the fourth, which actually runs **named**(8). The rest is icing: that `if` statement checks to see if there's actually a **named** program where it's expected to be, and the **echo** line reports that **named** is being started when the system is booting. You'll find that most of the servers started from `rc.inet2` are run from within blocks like these; simple tests to see if there's any obvious reason they shouldn't be run, a report that they're being started, and then the commands to start the services themselves. Again, `rc.inet2` is pretty heavily commented; poke around in it for a while.

NFS (Network File System)

The Network File System is used, obviously, to share files between machines on a network. The cool part about NFS is that it's designed such that one machine can mount shares from another transparently, and treat them like local files.

A couple of things have to happen for this to take place, though. First is the appropriate services have to be running on the server machine: these are **portmap(8)**, **nfsd(8)**, and **mountd(8)**. Second is the server has to explicitly `export` a filesystem tree to the client, which is accomplished via `exports(5)` file in `/etc`.

The first part of the equation is handled by installing the `tcpip1.tgz` package (from the N series) and letting `rc.inet2` do its thing. `/etc/exports` is a bit more fun.

Suppose I have a directory of images on `battlecat.tdn` that I want to mount on `ninja.tdn`. On `battlecat`, I'll need a line in `/etc/exports` that looks something like this:

```
/var/media/images    ninja.tdn(ro)
```

Then on `ninja`, I can simply

```
# mount -t nfs battlecat.tdn:/var/media/images /mnt
```

to mount the `images` directory as `/mnt` locally. Unfortunately, I've forbidden myself from writing to the shared directory-- that `(ro)` bit in `battlecat's /etc/exports` line is an option meaning `read-only`. Any such options need to exist after the client's name, inside parentheses, in a comma-separated list. For instance:

```
/var/media/images    ninja.tdn(rw,no_root_squash)
```

The `rw` is obviously `read-write` -- subject to user and group id mapping (see the `exports(5)` man page for an explanation), users on `ninja` are allowed to write to the shared directory. I don't like `squash`, so I think I'll leave it up to that man page to explain that one to you; if you plan on doing much with NFS, `exports(5)` will be your best friend. Might as well get comfy, right?

tcp_wrappers

tcp_wrappers is a basic system for preventing (and explicitly allowing) access to services from specified hosts. In a nutshell, it works like this:

inetd (the internet super-server) runs a lot of servers, many of which are wrapped by **tcpd**. In other words, **tcpd** is what actually runs these servers, but **inetd** doesn't know that (or care, really). **tcpd** logs the attempt to connect and then checks the files `/etc/hosts.allow` and `/etc/hosts.deny` files to see whether the connection should be allowed.

The rules contained in these files can be somewhat complex, but let's suppose `pyramid.tdn` is being really obnoxious and won't leave poor little `mojo.tdn` alone. `mojo.tdn` might throw a line into `/etc/hosts.deny` that looks like this:

```
ALL: pyramid.tdn
```

This line should be pretty clear: it prevents `pyramid` from using all of the services on `mojo` that are protected by `tcpd`. Were I to be annoyed by an entire domain in addition to `pyramid`, I could make that line read:

```
ALL: pyramid.tdn, .annoying.domain
```

But wait! My pal Hobbes is stuck with a machine on `.that.annoying.domain`, but I want him to be able to access me (just not the rest of his annoying friends). That's simple enough. Leaving `hosts.deny` as it stands, the following line in `hosts.allow` will let Hobbes in:

```
ALL: hobbes.annoying.domain
```

For much more detail, see **tcpd**(8), `hosts_access`(5), and `hosts_options`(5). The `tcp_wrappers` system is much more flexible than this, and is well worth checking out in more depth.

Summary

In this chapter, you learned how to configure your system to attach it to a network, how to tweak the configuration files, and some basic security principles. In addition, you learned what the Network File System is and how to get it working on your system. Making your system a part of a network allows you to access all sorts of resources like mail, news, and websites. See [Chapter 13](#) for information about how to use some basic network programs.

Chapter 6. The X Window System

Table of Contents

[xf86config](#)[XF86Setup](#)[Session Configuration Files](#)[Servers and Window Managers](#)[Selecting a Desktop](#)[Exporting displays](#)[Summary](#)

The X Window System is the standard GUI for all UNIX platforms, and this includes Linux. Unlike Windows and MacOS, in Linux and Unix the GUI is separate from the main operating system kernel. This adds stability to the system: if the GUI crashes, it doesn't take out the entire system.

One problem with X is that it has traditionally been very hard to configure for a system. However, Slackware 7 introduced a configure-less X setup that uses the framebuffer driver. This means that you don't have to go through the procedures described in [the section called *xf86config*](#) and [the section called *XF86Setup*](#). The framebuffer will also work on any VESA 2.0-compliant video cards. This means that just about any modern video card will work under X. However, the framebuffer is noticeably slower than using an X configuration tailored to your system.

If you choose to use the framebuffer server, you should install the `xxfb.tgz` package from the X software series. You should also choose one of the console resolutions during the configuration section of the installer. The recommended option for X is probably best for most people.

If you choose to configure X for your system, you'll need to follow the instructions in [the section called *xf86config*](#) or [the section called *XF86Setup*](#). The first section describes using **xf86config**(1), a commandline-based program for configuring X. The second section describes **XF86Setup**(1), a graphical version of the configuration program.

xf86config

xf86config is one of two programs that can be used to configure X on your system. The basic idea is simple: you will be presented with a series of questions and multiple choice answers. Choose the answer that best fits your system. After proceeding through the entire program, the `/etc/XF86Config(5)` file will be written and you will be ready to use X. If you mess up at some point, you'll have to kill the program using **control-c** and start over from the beginning.

It helps to know as much as possible about your monitor and video card before using **xf86config**. You can get information about your video card using the **SuperProbe** program:

```
# SuperProbe
```

This will give you a warning about possibly locking up the system. If this scares you off, quit using **control-c** before the five second time limit is up. Otherwise, you'll get some information about your video setup:

```
First video: Super-VGA
Chipset: ATI 264GT3 (3D Rage Pro) (Port Probed)
Memory: 4096 Kbytes
RAMDAC: ATI Mach64 integrated 15/16/24/32-bit
          DAC w/ clock
          (with 8-bit wide lookup tables)
          (programmable for 6/8-bit wide lookup tables)
Attached graphics coprocessor:
          Chipset: ATI Mach64
          Memory: 4096 Kbytes
```

That's what the information for an ATI Rage Pro video card looks like. Write down the information for your card, or switch over to another virtual terminal (using the **alt-function key** combinations) and run **xf86config** from there. You'll need your video card's information later. **xf86config** must be run as root, since it will be writing files and making symbolic links in places only root is allowed to:

```
# xf86config
```

Once you start **xf86config**, it will present a screen full of text telling you what it is going to do. Remember, there is no way to go back to the previous screen if you make a mistake, so pick carefully. Otherwise, you might have to do this a few times. Press **enter**, like it prompts you to do.

Mouse protocol

```
First specify a mouse protocol type. Choose one from the following list:
```

1. Microsoft compatible (2-button protocol)
2. Mouse Systems (3-button protocol)
3. Bus Mouse
4. PS/2 Mouse
5. Logitech Mouse (serial, old type, Logitech protocol)
6. Logitech MouseMan (Microsoft compatible)
7. MM Series
8. MM HitTablet
9. Microsoft IntelliMouse
10. Acecad tablet

```
If you have a two-button mouse, it is most likely of type 1, and if you have
a three-button mouse, it can probably support both protocol 1 and 2. There are
two main varieties of the latter type: mice with a switch to select the
protocol, and mice that default to 1 and require a button to be held at
boot-time to select protocol 2. Some mice can be convinced to do 2 by sending
a special sequence to the serial port (see the ClearDTR/ClearRTS options).
```

```
Enter a protocol number: █
```

Choose the kind of mouse you have from the list. These days, most mice will be PS/2 or a Microsoft Intellimouse. Older mice will probably require one of the other types listed.

Emulate3Buttons

1. Microsoft compatible (2-button protocol)
2. Mouse Systems (3-button protocol)
3. Bus Mouse
4. PS/2 Mouse
5. Logitech Mouse (serial, old type, Logitech protocol)
6. Logitech MouseMan (Microsoft compatible)
7. MM Series
8. MM HitTablet
9. Microsoft IntelliMouse
10. Acecad tablet

If you have a two-button mouse, it is most likely of type 1, and if you have a three-button mouse, it can probably support both protocol 1 and 2. There are two main varieties of the latter type: mice with a switch to select the protocol, and mice that default to 1 and require a button to be held at boot-time to select protocol 2. Some mice can be convinced to do 2 by sending a special sequence to the serial port (see the ClearDTR/ClearRTS options).

Enter a protocol number: 4

If your mouse has only two buttons, it is recommended that you enable Emulate3Buttons.

Please answer the following question with either 'y' or 'n'.
Do you want to enable Emulate3Buttons? y

If your mouse has only two buttons on it, you can choose to emulate a third button. Clicking both the left and right mouse buttons at the same time will be interpreted as a third button click. As many programs will make use of a third button, enabling it is recommended. If you have a three button mouse, this won't do anything.

Mouse device name

6. Logitech MouseMan (Microsoft compatible)
7. MM Series
8. MM HitTablet
9. Microsoft IntelliMouse
10. Acecad tablet

If you have a two-button mouse, it is most likely of type 1, and if you have a three-button mouse, it can probably support both protocol 1 and 2. There are two main varieties of the latter type: mice with a switch to select the protocol, and mice that default to 1 and require a button to be held at boot-time to select protocol 2. Some mice can be convinced to do 2 by sending a special sequence to the serial port (see the ClearDTR/ClearRTS options).

Enter a protocol number: 4

If your mouse has only two buttons, it is recommended that you enable Emulate3Buttons.

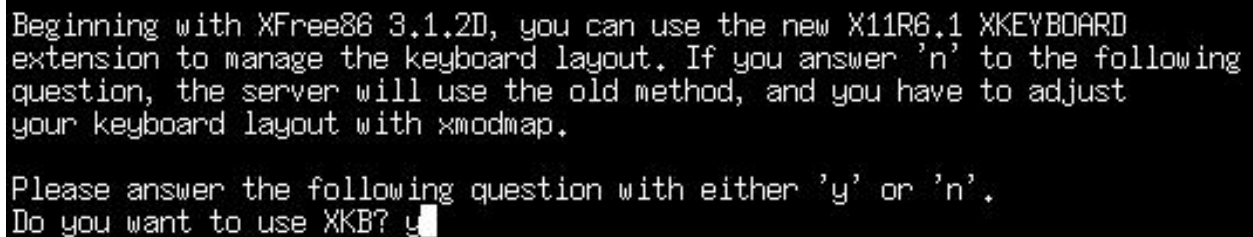
Please answer the following question with either 'y' or 'n'.
Do you want to enable Emulate3Buttons? y

Now give the full device name that the mouse is connected to, for example /dev/tty00. Just pressing enter will use the default, /dev/mouse.

Mouse device: /dev/mouse

The default of /dev/mouse is usually acceptable. However, if you have your mouse plugged into some strange port, you might need to change this. For most serial mice and PS/2 mice, the default is fine.

XKEYBOARD extension



```
Beginning with XFree86 3.1.2D, you can use the new X11R6.1 XKEYBOARD
extension to manage the keyboard layout. If you answer 'n' to the following
question, the server will use the old method, and you have to adjust
your keyboard layout with xmodmap.

Please answer the following question with either 'y' or 'n'.
Do you want to use XKB? y
```

You will probably want to use the X keyboard extensions. Not choosing this will cause some strange behavior with the backspace and delete keys. Choosing the keyboard extensions will cause keys to behave like they should.

Bindings for alt keys

If you want to enter characters from different languages, you should enable the bindings for alt keys.

If you're only going to be typing in English, you don't need to enable these bindings.

Horizontal sync range

You must indicate the horizontal sync range of your monitor. You can either select one of the predefined ranges below that correspond to industry-standard monitor types, or give a specific range.

It is VERY IMPORTANT that you do not specify a monitor type with a horizontal sync range that is beyond the capabilities of your monitor. If in doubt, choose a conservative setting.

```
hsync in kHz; monitor type with characteristic modes
1 31.5; Standard VGA, 640x480 @ 60 Hz
2 31.5 - 35.1; Super VGA, 800x600 @ 56 Hz
3 31.5, 35.5; 8514 Compatible, 1024x768 @ 87 Hz interlaced (no 800x600)
4 31.5, 35.15, 35.5; Super VGA, 1024x768 @ 87 Hz interlaced, 800x600 @ 56 Hz
5 31.5 - 37.9; Extended Super VGA, 800x600 @ 60 Hz, 640x480 @ 72 Hz
6 31.5 - 48.5; Non-Interlaced SVGA, 1024x768 @ 60 Hz, 800x600 @ 72 Hz
7 31.5 - 57.0; High Frequency SVGA, 1024x768 @ 70 Hz
8 31.5 - 64.3; Monitor that can do 1280x1024 @ 60 Hz
9 31.5 - 82.0; Monitor that can do 1280x1024 @ 76 Hz
10 31.5 - 95.0; Monitor that can do 1280x1024 @ 85 Hz
11 Enter your own horizontal sync range
```

Enter your choice (1-11): 4

This is the first of the questions relating to the monitor. It is important that you choose wisely here. Don't pick a range that is outside of your monitor's specifications. This is less important on newer monitors, as they will not attempt to do something outside of their specifications. Older monitors might be damaged, though. When in doubt, pick a conservative range.

Having your monitor's documentation would be a good reference for these next few questions. For most newer monitors, you can probably pick 31.5-48.5 or 31.5-57.0. Those of you will high-end monitors can choose one of the larger ranges. Or, you can enter your own horizontal sync range if you don't see one that fits quite right.

Vertical sync range

```
2 31.5 - 35.1; Super VGA, 800x600 @ 56 Hz
3 31.5, 35.5; 8514 Compatible, 1024x768 @ 87 Hz interlaced (no 800x600)
4 31.5, 35.15, 35.5; Super VGA, 1024x768 @ 87 Hz interlaced, 800x600 @ 56 Hz
5 31.5 - 37.9; Extended Super VGA, 800x600 @ 60 Hz, 640x480 @ 72 Hz
6 31.5 - 48.5; Non-Interlaced SVGA, 1024x768 @ 60 Hz, 800x600 @ 72 Hz
7 31.5 - 57.0; High Frequency SVGA, 1024x768 @ 70 Hz
8 31.5 - 64.3; Monitor that can do 1280x1024 @ 60 Hz
9 31.5 - 82.0; Monitor that can do 1280x1024 @ 76 Hz
10 31.5 - 95.0; Monitor that can do 1280x1024 @ 85 Hz
11 Enter your own horizontal sync range
```

Enter your choice (1-11): 4

You must indicate the vertical sync range of your monitor. You can either select one of the predefined ranges below that correspond to industry-standard monitor types, or give a specific range. For interlaced modes, the number that counts is the high one (e.g. 87 Hz rather than 43 Hz).

- 1 50-70
- 2 50-90
- 3 50-100
- 4 40-150
- 5 Enter your own vertical sync range

Enter your choice: 2

Once again, you'll need to know the specifications of your monitor to answer this question. When in doubt, go for a narrow range. A safe choice would probably be 50-90 or 50-100. If you don't see one that fits your monitor, you can choose to enter your own range.

Identification strings

You are now presented with three questions asking for identification strings for your monitor. These are not terribly important. You can just hit enter for all three of them if you'd like. Or, you can name them with whatever you want. These strings will be used in the configuration file for identification purposes.

Video card database

0	2 the Max MAXColor S3 Trio64V+	S3 Trio64V+
1	3DLabs Oxygen GMX	PERMEDIA 2
2	3DVision-i740 AGP	Intel 740
3	3Dlabs Permedia2 (generic)	PERMEDIA 2
4	928Movie	S3 928
5	ABIT G740 8MB SDRAM	Intel 740
6	AGP 2D/3D V. 1N, AGP-740D	Intel 740
7	AGX (generic)	AGX-014/15/16
8	ALG-5434(E)	CL-GD5434
9	AOpen AGP 2X 3D Navigator PA740	Intel 740
10	AOpen PA2010	Voodoo Banshee
11	AOpen PA45	SiS6326
12	AOpen PA50D	SiS6326
13	AOpen PA50E	SiS6326
14	AOpen PA50V	SiS6326
15	AOpen PA80/DVD	SiS6326
16	AOpen PG128	S3 Trio3D
17	AOpen PG975	3dimage975

Enter a number to choose the corresponding card definition.
Press enter for the next page, q to continue configuration.

The next large section of configuring X deals with your video card. Your card's documentation and the information from **SuperProbe** will both prove handy right now. You will want to look through the card database to pick out your card, so answer **y** to the question. Just hitting enter will take you past the card database and into the next section.

There are over 800 cards in the database. The left column contains a number for each card and the card's name. The right column contains the chipset for that card. Keep hitting enter until you find your video card in the list. When you've found your card, enter the number and hit enter. If you don't know what kind of video card you have, there are a few options. First, you could look at the **Chipset** line from **SuperProbe** and look for a card matching that chipset in the database. Or, you could use a generic SVGA card type. Many cards that do not have their own server are supported by the SVGA server, so this could be a safe choice.

After picking a card, it will give you some more information. Following the ATI Rage Pro example from above, it would give this kind of information:

Your selected card definition:

```
Identifier: ATI Mach64
Chipset:    ATI-Mach64
Server:     XF86_Mach64
Do NOT probe clocks or use any Clocks line.
```

At this point, you should check to make sure that you installed the server package. The XF86_Mach64 server is in the xma64.tgz package. Make sure that the correct server package is installed, or X will not be able to run.

Which server to run?

Now you must determine which server to run. Refer to the manpages and other documentation. The following servers are available (they may not all be installed on your system):

- 1 The XF86_Mono server. This is a monochrome server that should work on any VGA-compatible card, in 640x480 (more on some SVGA chipsets).
- 2 The XF86_VGA16 server. This is a 16-color VGA server that should work on any VGA-compatible card.
- 3 The XF86_SVGA server. This is a 256 color SVGA server that supports a number of SVGA chipsets. On some chipsets it is accelerated or supports higher color depths.
- 4 The accelerated servers. These include XF86_S3, XF86_Mach32, XF86_Mach8, XF86_8514, XF86_P9000, XF86_AGX, XF86_W32, XF86_Mach64, XF86_I128 and XF86_S3V.

These four server types correspond to the four different "Screen" sections in XF86Config (vga2, vga16, svga, accel).

Which one of these screen types do you intend to run by default (1-4)? █

This next question presents several servers that you can use. If you've selected your video card correctly, you can safely hit Enter. That will tell X to use the server that the card specified. Otherwise, you can pick to use the Mono server, VGA16 server, SVGA server, or an accelerated server. The best choice is to use the server that the card specified.

Setting the symbolic link

- VGA-compatible card, in 640x480 (more on some SVGA chipsets).
- 2 The XF86_VGA16 server. This is a 16-color VGA server that should work on any VGA-compatible card.
 - 3 The XF86_SVGA server. This is a 256 color SVGA server that supports a number of SVGA chipsets. On some chipsets it is accelerated or supports higher color depths.
 - 4 The accelerated servers. These include XF86_S3, XF86_Mach32, XF86_Mach8, XF86_8514, XF86_P9000, XF86_AGX, XF86_W32, XF86_Mach64, XF86_I128 and XF86_S3V.

These four server types correspond to the four different "Screen" sections in XF86Config (vga2, vga16, svga, accel).

Which one of these screen types do you intend to run by default (1-4)? 3

The server to run is selected by changing the symbolic link 'X'. For example, 'rm /usr/X11R6/bin/X; ln -s /usr/X11R6/bin/XF86_SVGA /usr/X11R6/bin/X' selects the SVGA server.

The directory /var/X11R6/bin exists. On many Linux systems this is the preferred location of the symbolic link 'X'. You can select this location when setting the symbolic link.

Please answer the following question with either 'y' or 'n'.

Do you want me to set the symbolic link? y █

Select y for setting the symbolic link. This will set up a link to the appropriate X server.

Video memory

```
Now you must give information about your video card. This will be used for
the "Device" section of your video card in XF86Config.
```

```
You must indicate how much video memory you have. It is probably a good
idea to use the same approximate amount as that detected by the server you
intend to use. If you encounter problems that are due to the used server
not supporting the amount memory you have (e.g. ATI Mach64 is limited to
1024K with the SVGA server), specify the maximum amount supported by the
server.
```

```
How much video memory do you have on your video card:
```

- 1 256K
- 2 512K
- 3 1024K
- 4 2048K
- 5 4096K
- 6 Other

```
Enter your choice: █
```

Select the amount of memory that your card has. **SuperProbe** can be used to give this information as well. If you've got something other than one of the choices, you can select `Other` and enter in a different amount. Make sure that you specify the amount of memory in kilobytes.

Identification strings

You'll be prompted to enter three more identification strings. These apply to your video card. As with the monitor strings, it is safe to hit enter for all three, unless you'd like to name your video card.

RAMDAC

You'll only need to choose a RAMDAC setting if you're using the S3, AGX, or W32 servers.

SuperProbe will tell you what kind of RAMDAC chip is present on your video card. Go through the list until you've found the correct chip, then enter the corresponding number. If you're not using the S3, AGX, or W32 servers, enter a `q` to continue without selecting a RAMDAC.

Clockchip setting

A Clockchip line in the Device section forces the detection of a programmable clock device. With a clockchip enabled, any required clock can be programmed without requiring probing of clocks or a Clocks line. Most cards don't have a programmable clock chip. Choose from the following list:

1	Chrontel 8391	ch8391
2	ICD2061A and compatibles (ICS9161A, DCS2824)	icd2061a
3	ICS2595	ics2595
4	ICS5342 (similar to SDAC, but not completely compatible)	ics5342
5	ICS5341	ics5341
6	S3 GenDAC (86C708) and ICS5300 (autodetected)	s3gendac
7	S3 SDAC (86C716)	s3_sdac
8	STG 1703 (autodetected)	stg1703
9	Sierra SC11412	sc11412
10	TI 3025 (autodetected)	ti3025
11	TI 3026 (autodetected)	ti3026
12	IBM RGB 51x/52x (autodetected)	ibm_rgb5xx

Just press enter if you don't want a Clockchip setting.
What Clockchip setting do you want (1-12)? █

If your card has a programmable clockchip, you'll want to select one from this next listing. Keep in mind that most cards do not have a programmable clockchip, so you should be safe by just hitting enter. **SuperProbe** should report if your card has a clockchip.

Clocks line

The next screenful of text talks about what a clocks line is. As it explains, you won't want one on most modern configurations. It will then prompt you, asking if it should probe for a clock. It will also tell you if the card needs to be probed or not. In the case of the ATI card, **xf86config** would say:

The card definition says to NOT probe clocks.

If it says something like that, choose **n** to the question of probing the card for clocks. Very old graphics cards will need to be probed. **xf86config** will tell you what needs to be done.

Video modes

```
For each depth, a list of modes (resolutions) is defined. The default
resolution that the server will start-up with will be the first listed
mode that can be supported by the monitor and card.
Currently it is set to:
```

```
"640x480" "800x600" "1024x768" "1280x1024" for 8bpp
"640x480" "800x600" "1024x768" "1280x1024" for 16bpp
"640x480" "800x600" "1024x768" "1280x1024" for 24bpp
"640x480" "800x600" "1024x768" for 32bpp
```

```
Note that 16, 24 and 32bpp are only supported on a few configurations.
Modes that cannot be supported due to monitor or clock constraints will
be automatically skipped by the server.
```

- 1 Change the modes for 8pp (256 colors)
- 2 Change the modes for 16bpp (32K/64K colors)
- 3 Change the modes for 24bpp (24-bit color, packed pixel)
- 4 Change the modes for 32bpp (24-bit color)
- 5 The modes are OK, continue.

```
Enter your choice: 5
```

Now it's time to select the video modes that your X server will use. You will see four different color depths - 8bpp, 16bpp, 24bpp, and 32bpp. Each will have a listing of the various video modes that can be run at that color depth. When you start up X, it will enter a default color depth and run at the first resolution listed for that color depth. If you'd like X to start up in a different resolution by default, now is the time to do that.

If the ordering of the video modes is fine, you can select OK, which will continue on with the configuration process. Otherwise, select the color depth that you'd like to change. For example, suppose you were presented with the following choices:

```
"640x480" "800x600" "1024x768" "1280x1024" for 8bpp
"640x480" "800x600" "1024x768" "1280x1024" for 16bpp
"640x480" "800x600" "1024x768" "1280x1024" for 24bpp
"640x480" "800x600" "1024x768" for 32bpp
```

If you'd like X to start up in a different resolution by default, you would first select a color depth to change. Then you would follow the directions given by **xf86config**. It will prompt you to enter the digits that correspond to the order of the resolutions. If you simply wanted to reverse the order of the resolutions, you could answer like so:

```
Which modes? 5432
```

This also allows you to delete resolutions. If your video card cannot run at 1280x1024, there's no reason to have it try. You could remove that line by answering with the following digits:

```
Which modes? 432
```

After selecting your modes at that color depth, you will be asked if you'd like a virtual screen that is bigger than the physical screen. A virtual screen is a screen that is bigger than the actual monitor. When you move the mouse around on the virtual screen, it will scroll a bit before coming to the edge. This allows you to fit more windows onto your monitor. However, because you will not be able to see everything at once, the virtual screen can be a bit annoying. It is still an interesting thing to play with, so you might want to try it out.

Then you'll be sent back to the list of video modes. After altering the video modes for the 24bpp color depth, it would look like this:

```
"640x480" "800x600" "1024x768" "1280x1024" for 8bpp
"640x480" "800x600" "1024x768" "1280x1024" for 16bpp
"1280x1024" "1024x768" "800x600" "640x480" for 24bpp
"640x480" "800x600" "1024x768" for 32bpp
```

Continue altering the video modes until you are satisfied with them. When you're done with this section, select **OK** to continue on.

Write the config file

At this point, configuration of X is complete. **xf86config** will ask if it should write the config file to `/etc/XF86Config`. If you'd like to be able to run X, you should answer **y** to that question, as that is where X will look for its configuration file.

Assuming that you answered all the questions correctly and have the X server package installed, you should now be able to start up X as follows:

```
$ startx
```

If you've installed KDE or GNOME, it should come up at this time. Otherwise, you might want to run **xwmconfig** and select the window manager that you want to use by default. Window managers will be described later on in this chapter. **xwmconfig** sets up the default window manager only for the user who runs it. If you have several users on your system, each will need to select their own window manager.

There are a few special keystrokes that might come in handy when using X. If you need to quit X at some point and you cannot close it down properly, there is a force-quit combination.

control-alt-backspace will kill X and dump you back to a command line. You can switch back to the command line terminals while running X by pressing **control-alt-function key**, which is similar to switching virtual terminals at the console. The X session is located on terminal 7, so you can get back into X with **alt-F7**. Finally, you can change video modes while running X.

control-alt-numeric keypad + will switch to the next highest resolution, while **control-alt-numeric keypad -** will switch to the next lowest resolution.

[Prev](#)
Summary

[Home](#)
[Up](#)

[Next](#)
XF86Setup

XF86Setup

The second way to configure X is to use **XF86Setup**, a graphical configuration program that comes as part of the `xset.tgz` package. You'll also need to install the `xvgl6.tgz` package.

To run XF86Setup, log in as root and type:

```
# XF86Setup
```

If you've already got an `/etc/XF86Config` file (because you've already configured X), you'll be asked if you want to use the existing XF86Config file for defaults. Otherwise, it'll switch right into graphics mode.

Figure 6-1. The initial XF86Setup screen.



XF86Setup is very similar to **xf86config**. It asks the same kind of questions, but presents them in a graphical environment. If you are wondering what one of the questions means, refer to the previous section for information. **XF86Setup** has a lot of help as well, so you should not have any difficulties in figuring it out.

Session Configuration Files

xinitrc and ~/.xinitrc

xinit(1) is the program that actually starts X; it is called by **startx**(1), so you may not have noticed it (and probably don't really need to). Its configuration file, however, determines which programs (including and especially the window manager) are run when X starts up. **xinit** first checks your home directory for a **.xinitrc** file. If the file is found, it gets run; otherwise, **/var/X11R6/lib/xinit/xinitrc** (the systemwide default) is used. Here's a simple **xinitrc** file:

```
#!/bin/sh
# $XConsortium: xinitrc.cpp,v 1.4 91/08/22 11:41:34 rws Exp $

userresources=$HOME/.Xresources
usermodmap=$HOME/.Xmodmap
sysresources=/usr/X11R6/lib/X11/xinit/.Xresources
sysmodmap=/usr/X11R6/lib/X11/xinit/.Xmodmap

# merge in defaults and keymaps

if [ -f $sysresources ]; then
    xrbdb -merge $sysresources
fi

if [ -f $sysmodmap ]; then
    xmodmap $sysmodmap
fi

if [ -f $userresources ]; then
    xrbdb -merge $userresources
fi

if [ -f $usermodmap ]; then
    xmodmap $usermodmap
fi

# start some nice programs

twm &
xclock -geometry 50x50-1+1 &
xterm -geometry 80x50+494+51 &
xterm -geometry 80x20+494-0 &
exec xterm -geometry 80x66+0+0 -name login
```

All of those `if` blocks are there to merge in various configuration settings from other files. We'll get to `.Xresources` in just a moment, but `.Xmodmap` we're going to leave alone. The interesting part of the file is toward the end, where various programs are run. This X session will begin with the `twm(1)` window manager, a clock, and three terminals. Note the `exec` before the last `xterm`. What that does is replace the currently running shell (the one that's executing this `xinitrc` script) with that `xterm(1)` command. When the user quits that `xterm`, the X session will end.

To customize your X startup, copy the default `/var/X11R6/lib/xinit/xinitrc` to `~/.xinitrc` and edit it, replacing those program lines with whatever you like. The end of mine is simply:

```
# Start the window manager:
exec startkde
```

Note that there are several `xinitrc.*` files in `/var/X11R6/lib/xinit` that correspond to various window managers and GUIs. You can use any of those, if you like.

.Xresources and .Xdefaults

Many X programs use a system called the X Resource Database to get various user preferences (fonts, colours, etc.) This database is maintained via the `xrdb(1)` program, which you will likely never need to run directly. Instead, it is run in Slackware from the `xinitrc`. The file that `xinitrc` tells `xrdb` to source for options is `~/.Xresources`. `xrdb` will also load `~/.Xdefaults`, so either of these filenames will work. A minimal `.Xresources` file looks like this:

```
xterm*background: black
xterm*foreground: gray
xterm*scrollBar: true
xterm*font: -*-lucidatypewriter-*-r-*-15-*-*-*-*-*
```

These four lines specify configuration information for the `xterm` program. An X resource is listed as follows:

```
program*option: setting/value
```

Thus, the sample `.Xresources` above should be fairly self-explanatory. Don't be thrown off by the `font` line; X fonts are always specified that way.

Servers and Window Managers

The X Window System was originally designed to work transparently across a network. One large server would actually run the X programs, but they would get displayed on various clients machines elsewhere on the network. The ability to remotely display programs can be a great advantage. The main disadvantage to this networking concept is that it is less secure than running applications on the local machine, and it takes a lot of network bandwidth to do. This is discussed later on in [the section called *Exporting displays*](#).

Even when you're running X on your own machine, you are still dealing with the client-server model. The server is the video-card specific portion. When you configured X and told it what kind of video card you had, that was telling it what server program to use. The client portion is all other programs you run under X. A special client, called the window manager, is responsible for the look and feel of your particular X session. The window manager is discussed in detail later.

The job of a window manager is to handle drawing windows on the screen with programs in them, as well as handling input from the mouse and keyboard. The first window managers did that and not much else. Today's window managers are much more complicated programs and are customizable in just about any way imaginable. They have all sorts of fancy options that let your desktop look different from anyone else's.

Having several window managers really separates Linux from Windows on the desktop. Under Windows, you have the one basic windowing environment. Under Linux, you can run one of many different window managers, each with a different look and different features. Some people would call this a weakness, because there is no consistent look. However, most Linux users would call this a strength because you can configure your system as much as you want.

Selecting a Desktop

For years, Unix was used almost exclusively as the operating system for servers, with the exception of high-powered professional workstations. Only the technically inclined were likely to use a Unix-like operating system, and the user interface reflected this fact. GUIs tended to be fairly bare-bones, designed to run a few necessarily graphical applications like CAD programs and image renderers. Most file and system management was conducted at the command line. Various vendors (Sun Microsystems, Silicon Graphics, etc) were selling workstation installations with an attempt at cohesive look and feel, but the wide variety of GUI toolkits in use by developers led inevitably to the dissolution of the desktop's uniformity. A scrollbar might not look the same in two different applications. Menus might appear in different places. Programs would have different buttons and checkboxes. Colors ranged widely, and were generally hard-coded in each toolkit. As long as the users were primarily technical professionals, none of this mattered much.

With the advent of free Unix-like operating systems and the growing number and variety of graphical applications, X has recently gained a wide desktop user base. Most users, of course, are accustomed to the consistent look and feel provided by Microsoft's Windows or Apple's MacOS; the lack of such consistency in X-based applications became a barrier to its wider acceptance. In response, two open source projects have been undertaken: The K Desktop Environment, or KDE, and the GNU Network Object Model Environment, known as GNOME. Each has a wide variety of applications, from taskbars and file managers to games and office suites, written with the same GUI toolkit and tightly integrated to provide a uniform, consistent desktop.

The differences in KDE and GNOME are generally fairly subtle. They each look different from the other, because each uses a different GUI toolkit. KDE is based on the Qt library from Troll Tech AS, while GNOME uses GTK, a toolkit originally developed for The GNU Image Manipulation Program (or The GIMP, for short). As separate projects, KDE and GNOME each have their own designers and programmers, with different development styles and philosophies. The result in each case, however, has been fundamentally the same: a consistent, tightly integrated desktop environment and application collection. The functionality, usability, and sheer prettiness of both KDE and GNOME rival anything available on other operating systems.

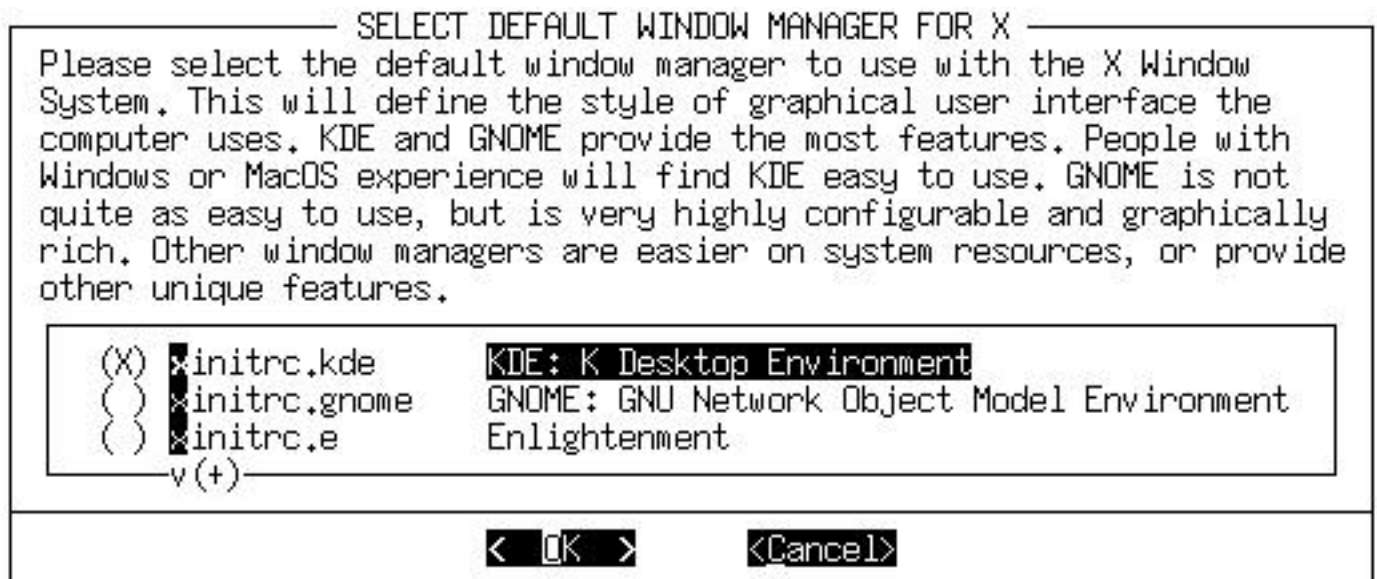
The best part, though, is that these advanced desktops are free. This means you can have either or both (yes, at the same time). The choice is yours.

In addition to the GNOME and KDE desktops, Slackware includes a large collection of window managers. Some are designed to emulate other operating systems, some for customization, others for speed. There's quite a variety. Of course you can install as many as you want, play with them all, and decide which you like the most.

To make desktop selection easy, Slackware also includes a program called **xwmconfig** that can be used to select a desktop or window manager. It is run like so:

```
$ xwmconfig
```

Figure 6-2. The Slackware `xwmconfig` program.



You'll be given a list of all the desktops and window managers installed. Just select the one you want from the list. Each user on your system will need to run this program, since different users can use different desktops, and not everyone will want the default one you selected at installation.

Then just start up X, and you're good to go:

```
$ startx
```

[Prev](#)

Servers and Window
Managers

[Home](#)

[Up](#)

[Next](#)

Exporting displays

Exporting displays

As was previously mentioned, it is possible to run X programs on one computer and display them on another. This is incredibly bandwidth-intensive, so you probably won't want to do this over a modem connection or over very long distances. Additionally, there are security considerations: exporting a display is not a very secure thing to do, since you'll be letting the entire network look at what you're doing. Still, it can be very useful on a local network.

An important thing to note here is the use of the words `client` and `server`. When exporting the display, you may become confused as to what's a client and what's a server. We refer to the machine that actually runs the X programs and sends the display information as the `server`. The machine that you use to display the remote program is called the `client`. When discussing the design of X, this is reversed. The program that displays things is called the `server`, while the running program is called the `client`. It's not too terribly confusing, but worth pointing out.

For this example, we'll be making use of two computers: `golf` is a fairly powerful server sitting underneath a desk on one side of a crowded room. It has a lot of RAM and a nice processor in it. In addition, it has lots of X programs on it, but no monitor. On the other side of the room is `couch`, an old machine with little RAM and not much disk. It is way too weak to run resource-intensive programs like Netscape. `couch` has two major advantages, though: it has a monitor, and it is sitting right next to the couch so you don't even have to get up to use it. Ideally, you would be able to run Netscape without getting off the couch. Exporting is the answer.

First, log in to `couch` and start up X. Then open your favorite terminal program (`xterm`, `rxvt`, `eterm`, `aterm`, or a host of others). The first step to remotely displaying X programs is to set up the client machine so that other machines are allowed to display to the machine. This uses the `xhost` program to control access. If you are on a secure internal network, you probably don't care who can remotely display programs. In that case, you would just let anyone on the network display:

```
couch$ xhost +  
access control disabled, clients can connect from any host
```

On the other hand, you might want to do this using machines that are on an insecure network (the Internet, a college network, or anything else that you don't have control over). You certainly don't want just anyone to connect. `xhost` allows you to be selective about who can display:

```
couch$ xhost + golf.foc  
golf.foc being added to access control list
```

Now, only `golf.foc` (the server mentioned earlier) can display programs to `couch`. You can see who has access to display programs by running `xhost` with no arguments:


```
couch$ xhost  
access control enabled, only authorized clients can connect  
INET:golf.foc  
INET:localhost  
INET:couch.foc  
LOCAL:
```

This is all you have to do to set up the client end of things. The next step is to set up the server so that it knows to display programs somewhere other than the monitor. Since the server doesn't have a monitor on it (and therefore doesn't have X running), it'll need to know where to display.

Setting up the server is not very difficult either. After connecting, you'll need to modify the `$DISPLAY` environment variable. By default, it will probably not be set to anything. You'll need to set `$DISPLAY` to the value of the remote host, plus a number that represents which X session to display to. You will almost always have only one X session running, so dealing with that variable shouldn't be an issue.

Here's how the `$DISPLAY` variable would be set on our example server using Bash as the shell. Other shells would use a different syntax, but the value should be the same.

```
golf$ export DISPLAY=couch.foc:0.0
```

That's all there is to setting up the server side of things. Now you just stay logged into the server and run X programs from there. All the screen output from the program will get sent across the network to the client machine, even though it's running on a computer across the room.

```
golf$ netscape &
```

This would run **netscape** off the server machine, but since the `DISPLAY` variable is set to couch, everything will get displayed there. You don't even have to get up to run those big X programs on your old terminal. One important note about this: the server machine will have to have all the X libraries and other support files needed to run the program. However, you won't need an X server or a `/etc/XF86Config` file, since nothing is getting displayed to the server.

Afterwards, you might want to disable display exporting by removing the server from your client's access control list:

```
couch$ xhost - golf.foc  
golf.foc being removed from access control list  
couch$
```

You can see how this a great way to share computing resources. But be careful, you may be the host of many X programs for many remote computers and not even know it.

Summary

In this chapter, you learned how to configure the X Window System using **xf86config** and **XF86Setup**. You should also know what a desktop environment and a window manager are, and how to switch around among the various choices. You should be able to export your X session to another computer. At this point, you should be up and running in a graphical environment.

Chapter 7. Booting

Table of Contents

[LILO](#)[LOADLIN](#)[Dual Booting](#)[Summary](#)

The process of booting your Linux system can sometimes be easy and sometimes be difficult. Many users install Slackware on their computer and that's it. They just turn it on and it's ready to use. Others, however, must use another operating system for certain tasks, so they need both operating systems available on the machine.

This section covers using LILO and Loadlin, the two booters included with Slackware. It also explains some typical dual booting scenarios and how you could go about setting it up.

LILO

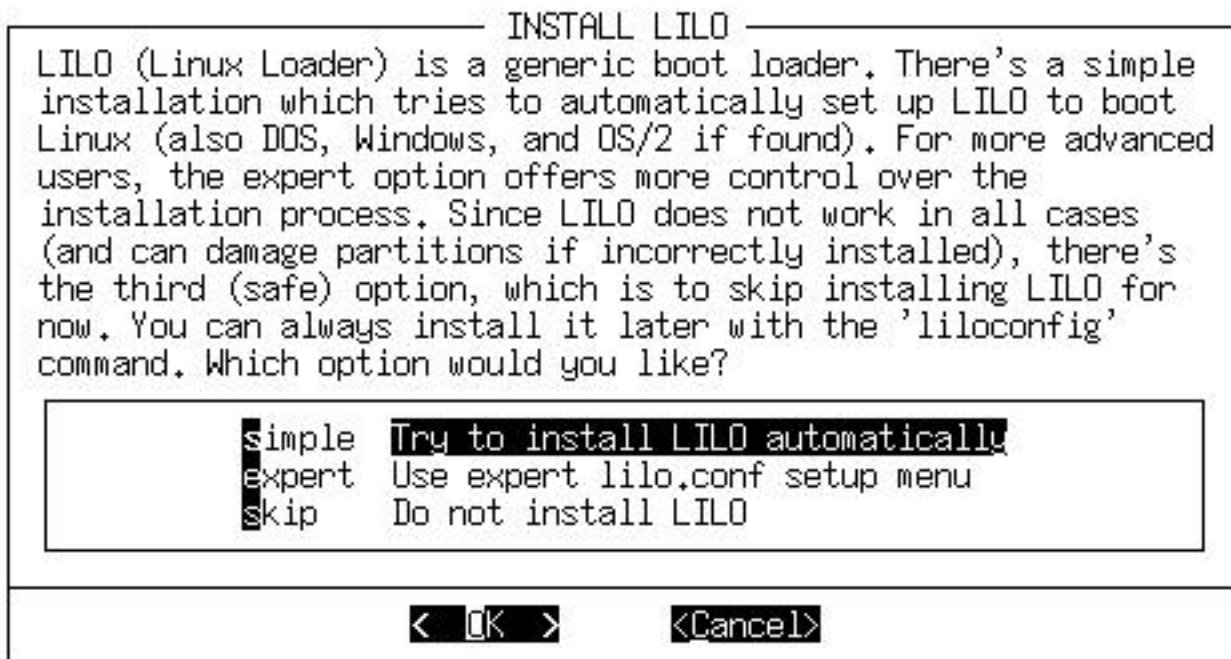
The Linux Loader, or LILO, is the most popular booter in use on Linux systems. It is quite configurable and can easily be used to boot other operating systems.

Slackware Linux comes with a menu-driven configuration utility called **liloconfig**. This program is first run during the setup process, but you can invoke it later by typing **liloconfig** at the prompt.

LILO reads its settings from the `/etc/lilo.conf(5)` file. It is not read each time you boot up, but instead is read each time you install LILO. LILO must be reinstalled to the boot sector each time you make a configuration change. **liloconfig** will help you build the configuration file so that you can install LILO for your system. If you prefer to edit `/etc/lilo.conf` by hand, then reinstalling LILO just involves type `/sbin/lilo` at the prompt.

When you first invoke **liloconfig**, it will look like this:

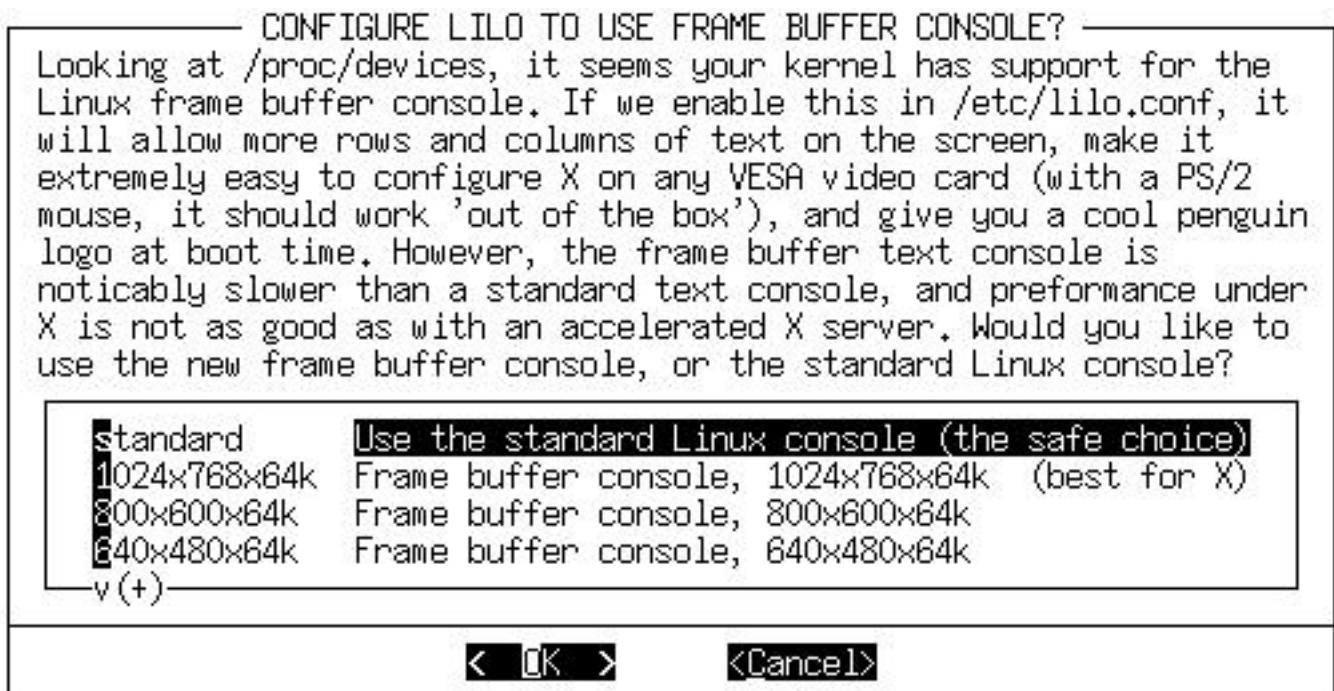
Figure 7-1. The initial liloconfig screen.



If this is your first time setting up LILO, you should pick `simple`. Otherwise, you might find `expert` to be faster if you are familiar with LILO and Linux. Selecting `simple` will begin the LILO configuration.

If kernel frame buffer support is compiled into your kernel, **liloconfig** will ask which video resolution you would like to use. This is the resolution that is also used by the XFree86 frame buffer server. If you do not want the console to run in a special video mode, selecting `normal` will keep the standard 80x25 text mode in use.

Figure 7-2. Liloconfig asking what video mode to use for the framebuffer.



The next part of the LILO configuration is selecting where you want it installed. This is probably the most important step. The list below explains the installation places:

Root

This option installs LILO to the beginning of your Linux root partition. This is the safest option if you have other operating systems on your computer. It ensures that any other booters are not overwritten. The disadvantage is that LILO will only load from here if your Linux drive is the first drive on your system.

Floppy

This method is even safer than the previous one. It creates a boot floppy that you can use to boot your Linux system. This keeps the booter off the hard disk entirely, so you only boot this floppy when you want to use Slackware.

MBR

You will want to use this method if Slackware is the only operating system on your computer, or if you will be using LILO to choose between multiple operating systems on your computer.



This option will overwrite any other booter you have in the MBR.

After selecting the installation location, **liloconfig** will write the configuration file and install LILO. That's it. If you select the **expert** mode you will receive a special menu. This menu allows you to tweak the `/etc/lilo.conf` file, add other operating systems to your boot menu, and set LILO to pass special kernel parameters at boot time. The expert menu looks like this:

Figure 7-3. liloconfig expert mode.



Whatever your system configuration is, setting up a working boot loader is easy. **liloconfig** makes setting it up a cinch. However, there are instances where LILO just won't work on a system. Fortunately, there are other options.

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

LOADLIN

LOADLIN

The other booting option that comes with Slackware Linux is LOADLIN. LOADLIN is a DOS executable that can be used to start Linux from a running DOS system. It requires the Linux kernel to be on the DOS partition so that LOADLIN can load it and properly boot the system.

During the installation process, LOADLIN will be copied to root's home directory as a .ZIP file. There is no automatic setup process for LOADLIN. You will need to copy the Linux kernel (/vmlinuz) and the LOADLIN file from root's home directory to the DOS partition.

LOADLIN is useful if you would like to make a boot menu on your DOS partition. A menu could be added to your AUTOEXEC.BAT file that would allow you to pick between Linux or DOS. A choice of Linux would run LOADLIN, thus booting your Slackware system. This AUTOEXEC.BAT file under Windows 95 will provide a sufficient boot menu:

```
@ECHO OFF
SET PROMPT=$P$G
SET PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\
CLS
ECHO Please Select Your Operating System:
ECHO.
ECHO [1] Slackware Linux
ECHO [2] Windows 95
ECHO.
CHOICE /C:12 "Selection? -> "
IF ERRORLEVEL 2 GOTO WIN
IF ERRORLEVEL 1 GOTO LINUX
:WIN
CLS
ECHO Starting Windows 95...
WIN
GOTO END
:LINUX
ECHO Starting Slackware Linux...
CD \LINUX
LOADLIN C:\LINUX\VMLINUZ ROOT=<root partition device> RO
GOTO END
:END
```

You will want to specify your root partition as a Linux device name, like /dev/hda2 or something else. You can always use LOADLIN at the command line. You simply use it in

the same manner as it is in the example above. The LOADLIN documentation comes with many examples on how to use it.

[Prev](#)

Booting

[Home](#)

[Up](#)

[Next](#)

Dual Booting

Dual Booting

Many users set up their computers to boot Slackware Linux and another operating system. We've described several typical dual boot scenarios below, in case you are having difficulty setting up your system.

Windows 9x/DOS

Setting up a computer with both Windows 9x and Linux is probably the most common dual boot scenario. There are numerous ways you can setup the booting, but this section will cover two.

Often times when setting up a dual boot system, a person will devise a perfect plan for where everything should go but mess up the installation order. It is very important to understand that operating systems need to be installed in a certain order for a dual boot setup to work. Linux always offers control over what, if anything, gets written to the Master Boot Record. Therefore, it's always advisable to install Linux last. Windows should be installed first, since it will always write its booter to the Master Boot Record.

Using LILO

Most people will want to use LILO to chose between Linux and Windows. As stated above, you should install Windows first, then Linux.

Let's say you have a 47GB IDE hard disk as the only drive in your system. Let's also say that you want to give half of that space to Windows and half of that space to Linux. This will present a problem when trying to boot Linux. I do not know the specific geometry of the drive, but chances are that 23.5GB down the drive will be well past the 1024th cylinder. A better layout for this drive would be:

```
1GB   Windows boot (C:)
1GB   Linux root (/)
22.5  Windows misc (D:)
22.5  Linux /usr (/usr)
```

You would also want to set aside an adequate amount of space for a Linux swap partition. The unwritten rule is to use twice the amount of RAM you have in disk space. A 64MB system would have 128MB of swap, and so on.

With your partitions layed out, you should proceed to install Windows. After that is set up and working, you should install Linux. The LILO installation needs special attention. You

will want to select the `expert` mode for installing LILO.

Begin a new LILO configuration. You will want to install it to Master Boot Record so that it can be used to choose between the two operating systems. From the menu, add your Linux partition and add your Windows (or DOS) partition. Once that's complete, you can install LILO.

Reboot the computer. LILO should load and wait for user interaction. You can press `Alt` to get the `boot :` prompt. Type the name of the operating system you want to load (these names were selected when you setup LILO). If you forgot the name, press **Tab** to get a list of operating systems that you can boot.

You can configure LILO even further by editing the `/etc/lilo.conf` file on your Linux partition. You can set it up to display a text menu, and always present the prompt. For example, if I wanted my LILO display to do this:

```
System Boot Menu
=====
1 - Linux
2 - Windows

LIL0 boot:
```

My `/etc/lilo.conf` file would look like this:

```
# LIL0 configuration file

boot = /dev/hda
vga = normal
message = /boot/message

image = /vmlinuz
    root = /dev/hda2
    label = 1
    read-only

other = /dev/hda1
    label = 2
    table = /dev/hda
```

And my `/boot/message` file would look like this:

```
System Boot Menu
=====
1 - Linux
2 - Windows
```

LILO is quite a configurable boot loader. It's not just limited to booting Linux or DOS. It can boot just about anything. The man pages for **lilo**(8) and **lilo.conf**(5) provide more detailed information.

What if LILO doesn't work? There are instances where LILO just won't work on a particular machine. Fortunately, there is another way to dual boot Linux and Windows.

Using LOADLIN

This method can be used if LILO doesn't work on your system, or if you just don't want to set up LILO. This method is also ideal for the user that reinstalls Windows often. Each time you reinstall Windows, it will overwrite the Master Boot Record, thus destroying any LILO installation. With LOADLIN, you are not subject to that problem. The biggest disadvantage is that you can only use LOADLIN to boot Linux.

With LOADLIN, you can install the operating systems in any order desired. Be careful about installing things to the Master Boot Record, you do not want to do that. LOADLIN relies on the Windows partition being bootable. So during the Slackware installation, make sure you skip the LILO setup.

After installing the operating systems, copy the `lodlinX.zip` (where `X` is a version number, such as `16a`) file from root's home directory to your Windows partition. Also copy your kernel image to the Windows partition. You will need to be in Linux for this to work. This example shows how to do this:

```
# mkdir /win
# mount -t vfat /dev/hda1 /win
# mkdir /win/linux
# cd /root
# cp loadlin* /win/linux
# cp /vmlinuz /win/linux
# cd /win/linux
# unzip loadlin16a.zip
```

That will create a `C:\LINUX` directory on your Windows partition (assuming it's `/dev/hda1`) and copy over the necessary stuff for LOADLIN. After doing this, you will need to reboot into Windows to setup a boot menu.

Once in Windows, get to a DOS prompt. First, we need to make sure the system is set to not boot into the graphical interface.

```
C:\>cd \
C:\>attrib -r -a -s -h MSDOS.SYS
C:\>edit MSDOS.SYS
```

Add this line to the file:

```
BootGUI=0
```

Now save the file and exit the editor. Now edit `C:\AUTOEXEC.BAT` so we can add a boot menu. The following provides an example of what a boot menu block in `AUTOEXEC.BAT`

would look like:

```
cls
echo System Boot Menu
echo.
echo 1 - Linux
echo 2 - Windows
echo.
choice /c:12 "Selection? -> "
if errorlevel 2 goto WIN
if errorlevel 1 goto LINUX
:LINUX
cls
echo "Starting Linux..."
cd \linux
loadlin c:\linux\vmlinux root=/dev/hda2 ro
goto END
:WIN
cls
echo "Starting Windows..."
win
goto END
:END
```

The key line is the one that runs LOADLIN. We tell it the kernel to load, the Linux root partition, and that we want it mounted read-only initially.

The tools for these two methods are provided with Slackware Linux. There are numerous other booters on the market, but these should work for most dual boot setups.

Windows NT

This is the second most common dual booting situation. Windows NT presents several more problems than dual booting between Windows 9x and Linux. The one we are most concerned with is if the Master Boot Record is overwritten with LILO, NT will not boot successfully. Therefore, we must use the OS Loader that comes with Windows NT. The steps below show how you should setup a Windows NT and Linux dual boot system.

1. Install Windows NT
2. Install Linux, making sure LILO is installed to the superblock of the Linux partition
3. Get the first 512 bytes of the Linux root partition and store it on the Windows NT partition
4. Edit C:\BOOT.INI under Windows NT to add a Linux option

Installing Windows NT should be fairly straightforward, as should installing Linux. From there, it gets a little more tricky. Grabbing the first 512 bytes of the Linux partition is easier

than it sounds. You will need to be in Linux to accomplish this. Assuming your Linux partition is `/dev/hda2`, issue this command:

```
# dd if=/dev/hda2 of=/tmp/bootsect.lnx bs=1 count=512
```

That's it. Now you need to copy `bootsect.lnx` to the Windows NT partition. Here's where we run into another problem. Linux does not have stable write support for the NTFS filesystem. If you installed Windows NT and formatted your drive as NTFS, you will need to copy this file to a FAT floppy and then read from it under Windows NT. If you formatted the Windows NT drive as FAT, you can simply mount it under Linux and copy the file over. Either way, you will want to get `/tmp/bootsect.lnx` from the Linux drive to `C:\BOOTSECT.LNX` on the Windows NT drive.

The last step is adding a menu option to the Windows NT boot menu. Under Windows NT open a command prompt.

```
C:\WINNT>cd \  
C:\>attrib -r -a -s -h boot.ini  
C:\>edit boot.ini
```

Add this line to the end of the file:

```
C:\bootsect.lnx="Slackware Linux"
```

Save the changes and exit the editor. When you reboot Windows NT, you will have a Linux option on the menu. Choosing it will boot into Linux.

Linux

Yes, people really do this. This is definitely the easiest dual boot scenario. You can simply use LILO and add more entries to the `/etc/lilo.conf` file. That's all there is to it.

[Prev](#)

LOADLIN

[Home](#)

[Up](#)

[Next](#)

Summary

Summary

This chapter discussed booting your system using either LILO or Loadlin. It also discussed booting between Linux and other operating systems. You should be able to configure your booting method properly and dual boot with another operating system, should you choose to.

IV. Using Slackware Linux

Table of Contents

- 8. [The Shell](#)
- 9. [Filesystem Structure](#)
- 10. [Handling Files and Directories](#)
- 11. [Process Control](#)
- 12. [Essential System Administration](#)
- 13. [Basic Network Commands](#)
- 14. [Archive Files](#)
- 15. [vi](#)
- 16. [Slackware Package Management](#)
- 17. [ZipSlack and BigSlack](#)

Chapter 8. The Shell

Table of Contents

[Users](#)[The Command Line](#)[The Bourne Again Shell \(bash\)](#)[Virtual Terminals](#)[Summary](#)

In a graphical environment, the interface is provided by a program that creates windows, scrollbars, menus, etc. In a commandline environment, the user interface is provided by a shell, which interprets commands and generally makes things useable. Immediately after logging in (which is covered in this chapter), users are put into a shell and allowed to go about their business. This chapter serves as an introduction to the shell, and to the most common shell among Linux users-- the Bourne Again Shell (bash). For more detailed information on anything in this chapter, check out the **bash**(1) man page.

Users

Logging In

So you've booted, and you're looking at something that looks like this:

```
Welcome to Linux 2.2.14
darkstar login:
```

Hmm.. nobody said anything about a login. And what's a darkstar? Don't worry; you probably didn't accidentally fire up a hyperspace comm-link to the Empire's artificial moon. (I'm afraid the hyperspace comm-link protocol isn't currently supported by the Linux kernel.) No, darkstar is just the name of one of our computers, and its name gets stamped on as the default. If you specified a name for your computer during setup, you should see it instead of darkstar.

As for the login... If this is your first time, you'll want to log in as **root**. You'll be prompted for a password; if you set one during the setup process, that's what it's looking for. If not, just hit enter. That's it-- you're in!

Root: The Superuser

Okay, who or *what* is root ? And what's it doing with an account on *your* system?

Well, in the world of Unix and similar operating systems (like Linux), there are users and then there are users. We'll go into this in more detail later, but the important thing to know now is that root is the user above all users; root is all-powerful and all-knowing, and *nobody* disobeys root. It just isn't allowed. Root is what we call a superuser , and rightly so. And best of all, root is *you*.

Cool, huh?

If you're not sure: yes, that's very cool. The catch is, though, that root is inherently allowed to break anything it so desires. You might want to skip ahead to [Chapter 12](#) and see about adding a user; then login as that user and work from there. The traditional wisdom is that it's best to only become superuser when absolutely necessary, so as to minimize the possibility of accidentally breaking something.

By the way, if you decide you want to be root while you're logged in as someone else, no problems. Just use the **su(1)** command. You'll be asked for root's password and then it will make you root until you **exit** or **logout**. You can also become any other user using **su**, provided you know that user's password: **su logan**, for instance, would make you me.

[Prev](#)

Using Slackware Linux

[Home](#)

[Up](#)

[Next](#)

The Command Line

The Command Line

Running Programs

It's hard to get much accomplished without running a program; you might be able to prop something up with your computer or hold a door open, and some will make the most lovely humming noise when running, but that's really about it. And I think we can all agree that its use as a humming doorstop isn't what brought the personal computer the popularity it now enjoys.

So, remember how almost everything in Linux is a file? Well, that goes for programs, too. Every command you run (that isn't built into the shell) resides as a file somewhere. You run a program simply by specifying the full path to it.

For instance, remember that **su** command from the last section? Well, it's actually in the `/bin` directory: **/bin/su** would run it nicely.

So why, then, does just typing **su** work? After all, you didn't say it was in `/bin`. It could just as easily have been in `/usr/local/share`, right? How did it *know*? The answer to that lies in the `PATH` environment variable; most shells have either `PATH` or something very like `PATH`. It basically contains a list of directories to look in for programs you try to run. So when you ran **su**, your shell ran through its list of directories, checking each one for an executable file called **su** that it could run; the first one it came to, it ran. This happens whenever you run a program without specifying a full path to it; if you get a `Command not found` error, that only means that the program you tried to run isn't in your `PATH`. (Of course, this would be true if the program doesn't exist at all...) We'll discuss environment variables in more depth in [the section called *The Bourne Again Shell \(bash\)*](#).

Remember also that `.` is shorthand for the directory I'm in, so if you happened to be in `/bin`, **./su** would have worked as an explicit full path.

Wildcard Matching

Nearly every shell recognizes some characters as being substitutes or abbreviations that mean anything goes here. Such characters are aptly named *wildcards*; the most common are `*` and `?`. By convention, `?` usually matches any single character. For instance, suppose you're in a directory with three files: `ex1.txt`, `ex2.txt`, and `ex3.txt`. You want to copy all of those files (using the **cp** command we cover in [the section called *cp* in Chapter 10](#)) to another directory, say `/tmp`. Well, typing **cp ex1.txt ex2.txt ex3.txt /tmp** is entirely too much work. It's much easier to type **cp ex?.txt /tmp**; the `?` will match each of the characters `1`, `2`, and `3`, and each in turn will be substituted in.

What's that you say? That's *still* too much work? You're right. It's appalling; we have labor laws to protect us from that sort of thing. Fortunately, we also have `*`. As was already mentioned, `*` matches any number of characters, including `.`. So if those three files were the only ones in the directory, we could have simply said `cp */tmp` and gotten them all in one fell swoop. Suppose, though, that there is also a file called `example.txt` and one called `hejaz.txt`. We want to copy `example.txt` but not `hejaz.txt`; `cp example* /tmp` will do that for us.

`cp example?.txt /tmp`, would, of course, only get our original three files; there's no character in `example.txt` to match that `?`, so it would be left out.

Input/Output Redirection and Piping

(Here comes something cool.)

```
$ ps > blargh
```

Y'know what that is? That's me running `ps` to see which processes are running; `ps` is covered in [Chapter 11](#). That's not the cool part. The cool part is `> blargh`, which means, roughly, take the output from `ps` and write it to a file called `blargh`. But wait, it gets cooler.

```
$ ps | less
```

That one takes the output from `ps` and pipes it through `less`, so I can scroll through it at my leisure.

```
$ ps >> blargh
```

This is the third most commonly used redirector; it does the same thing as `>`, except that `>>` will append output from `ps` to the file `blargh`, if said file exists. If not, just like `>`, it will be created. (`>` will obliterate the current contents of `blargh`.)

There is also a `<` operator, which means take your input from the following, but it's not used nearly so often.

```
$ fromdos < dosfile.txt > unixfile.txt
```

Redirection gets really fun when you start piling it up:

```
$ ps | tac >> blargh
```

That will run `ps`, reverse the lines of its output, and append those to the file `blargh`. You can stack as many of these up as you want; just be careful to remember that they get interpreted from left to right.

See the `bash(1)` man page for more detailed information on redirection.

The Bourne Again Shell (bash)

Environment Variables

A Linux system is a complex beast, and there's a lot to keep track of, a lot of little details that come into play in your normal interactions with various programs (some of which you might not even need to be aware of). Nobody wants to pass a bunch of options to every program that gets run, telling it what kind of terminal is being used, the hostname of the computer, how their prompt should look...

So as a coping mechanism, users have what's called an environment. The environment defines the conditions in which programs run, and some of this definition is variable; the user can alter and play with it, as is only right in a Linux system. Pretty much any shell will have environment variables (if not, it's probably not a very useable shell). Here we will give an overview of the commands bash provides for manipulating its environment variables.

```
$ set
```

set by itself will show you all of the environment variables that are currently defined, as well as their values. Like most **bash** built-ins, it can also do several other things (with parameters); we'll leave it to the **bash(1)** man page to cover that, though. An excerpt from a **set** command on one of my computers looks like this:

```
PATH=/usr/local/lib/qt/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/usr/openwin/bin:/usr/games:./usr/local/ssh2/bin:/usr/local/ssh1/bin:/usr/share/texmf/bin:/usr/local/sbin:/usr/sbin:/home/logan/bin
PIPESTATUS=( [0]="0" )
PPID=4978
PS1='\h:\w\$ '
PS2='> '
PS4='+ '
PWD=/home/logan
QTDIR=/usr/local/lib/qt
REMOTEHOST=ninja.tdn
SHELL=/bin/bash
```

Notice that **PATH** variable we discussed earlier; I can run anything in any of those directories simply by typing its base filename.

```
$ unset VARIABLE
```

unset will remove any variables that you give it, wiping out both the variable and its value; **bash** will forget that variable ever existed. (Don't worry. Unless it's something you explicitly defined in that shell session, it'll probably get redefined in any other session.)

```
$ export VARIABLE=some_value
```

Now, **export** is truly handy. Using it, you give the environment variable **VARIABLE** the value **some_value**; if **VARIABLE** didn't exist, it does now. If **VARIABLE** already had a value, well, it's gone. That's not so good, if you're just trying to add a directory to your **PATH**. In that case, you probably want to do something like this:

```
$ export PATH=$PATH:/some/new/directory
```

Note the use of **\$PATH** there: when you want **bash** to interpret a variable (replace it with its value), tack a **\$** onto the beginning of the variable's name. For instance, **echo \$PATH** will echo the value of **PATH**, in my case:

```
$ echo $PATH
/usr/local/lib/qt/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:
/usr/openwin/bin:/usr/games:./usr/local/ssh2/bin:/usr/local/ssh1/bin:
/usr/share/texmf/bin:/usr/local/sbin:/usr/sbin:/home/logan/bin
```

Tab Completion

(Here comes something cool again.)

1. A commandline interface means lots of typing.
2. Typing is work.
3. Nobody likes work.

From 3 and 2, we can determine that 4) nobody likes typing. Fortunately, **bash** saves us from 5 (nobody likes a commandline interface).

How does **bash** accomplish this wonderful feat, you ask? In addition to the wildcard expansion we discussed before, **bash** features tab completion .

Tab completion works something like this: You're typing the name of a file. Maybe it's in your `PATH`, maybe you're typing it out explicitly. All you have to do is type enough of the filename to uniquely identify it. Then hit the tab key. **bash** will figure out what you want and finish typing it for you!

Example time. `/usr/src` contains two subdirectories: `/usr/src/linux` and `/usr/src/sendmail`. I want to see what's in `/usr/src/linux`. So I just type **ls /usr/src/l**, hit the **TAB** key, and **bash** gives me `ls /usr/src/linux`.

Now, suppose there are two directories `/usr/src/linux` and `/usr/src/linux-old`; If I type **/usr/src/l** and hit **TAB**, **bash** will fill in as much as it can, and I'll get `/usr/src/linux`. I can stop there, or I can hit **TAB** again, and **bash** will show a list of directories that match what I've typed so far.

Hence, less typing (and hence, people can like commandline interfaces). I told you it was cool.

[Prev](#)

The Command Line

[Home](#)[Up](#)[Next](#)

Virtual Terminals

Virtual Terminals

So you're in the middle of working on something and you decide you need to do something else. You could just drop what you're doing and switch tasks, but this is a multi-user system, right? And you can log in as many times simultaneously as you want, right? So why should you have to do one thing at a time?

You don't. We can't all have multiple keyboards, mice, and monitors for one machine; chances are most of us don't want them. Clearly, hardware isn't the solution. That leaves software, and Linux steps up on this one, providing virtual terminals, or VTs.

By pressing Alt and a function key, you can switch between virtual terminals; each function key corresponds to one. Slackware has logins on 6 VTs by default. Alt+F2 will take you to the second one, Alt+F3 to the third, etc.

The rest of the function keys are reserved for X sessions. Each X session uses its own VT, beginning with the seventh (Alt+F7) and going up. When in X, the Alt+Function key combination is replaced with Ctrl+Alt+Function; so if you are in X and want to get back to a text login (without exiting your X session), Ctrl+Alt+F3 will take you to the third. (Alt+F7 will take you back, assuming you're using the first X session.)

Summary

This chapter discussed users, the shell, the command line, and virtual terminals. You should feel comfortable working on the command line, running programs, and using pipes and redirection operators to combine commands. Finally, you should have some idea of the power of the root user and why always running as root is a bad thing.

Chapter 9. Filesystem Structure

Table of Contents

[Ownership](#)[Permissions](#)[Links](#)[Mounting Devices](#)[NFS Mounts](#)[Summary](#)

We have already discussed the directory structure in Slackware Linux. You are able to find files and directories that you need. But there is more to the filesystem than just the directory structure.

Linux is a multiuser operating system. Every aspect of the system is multiuser, even the filesystem. The system stores information like who owns a file and who can read it. There are other unique parts about the filesystems, such as links and NFS mounts. This section explains these, as well as the multiuser aspects of the filesystem.

Ownership

The filesystem stores ownership information for each file and directory on the system. This includes what owner and group own a particular file. The easiest way to see this information is with the **ls** command:

```
$ ls -l /usr/bin/wc
-rwxr-xr-x  1 root    bin      7368 Jul 30  1999 /usr/bin/wc
```

We are interested in the third and fourth columns. These contain the username and group name that owns this file. We see that the user `root` and the group `bin` own this file.

We can easily change the file owners with the **chown**(1) (which means `change owner`) and **chgrp**(1) (which means `change group`) commands. To change the file owner to `daemon`, we would use **chown**:

```
# chown daemon /usr/bin/wc
```

To change the group owner to `root`, we would use **chgrp**:

```
# chgrp root /usr/bin/wc
```

We can also use **chown** to specify the user and group owners for a file:

```
# chown daemon.root /usr/bin/wc
```

File ownership is a very important part of using a Linux system, even if you are the only user. You sometimes need to fix ownerships on files and device nodes.

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

Permissions

Permissions

Permissions are the other important part of the multiuser aspects of the filesystem. With these, you can change who can read, write, and execute files.

The permission information is stored as four octal digits, each specifying a different set of permissions. There are owner permissions, group permissions, and world permissions. The fourth octal digit is used to store special information such as set user ID, set group ID, and the sticky bit. The octal values assigned to the permission modes are (they also have letters associated with them that are displayed by programs such as **ls** and can be used by **chmod**):

Table 9-1. Octal Permission Values

Permission Type	Octal Value	Letter Value
"sticky" bit	1	t
set user ID	4	s
set group ID	2	s
read	4	r
write	2	w
execute	1	x

You add the octal values for each permission group. For example, if you want the group permissions to be read and write, you would use 6 in the group portion of the permission information.

bash's default permissions are:

```
$ ls -l /bin/bash
-rwxr-xr-x  1 root    bin  477692 Mar 21 19:57 /bin/bash
```

The first dash would be replaced with a **d** if this was a directory. The three permission groups (owner, group, and world) are displayed next. We see that the owner has read, write, and execute permissions (rwx). The group has only read and execute (r-x). And everyone else has only read and execute (r-x).

How would we set permissions on another file to resemble **bash**'s? First, let's make an example file:

```
$ touch /tmp/example
$ ls -l /tmp/example
-rw-rw-r--  1 david  users    0 Apr 19 11:21 /tmp/example
```

We will use **chmod**(1) (which means **change mode**) to set the permissions on the example file. Add the octal numbers for the permissions you want. For the owner to have read, write,

and execute, we would have a value of 7. Read and execute would have 5. Run those together and pass them to **chmod** like this:

```
$ chmod 755 /tmp/example
$ ls -l /tmp/example
-rwxr-xr-x  1 david  users    0 Apr 19 11:21 /tmp/example
```

To set special permissions, add the numbers together and place them in the first column. For example, to make it set user ID and set group ID, we use 6 as the first column:

```
$ chmod 6755 /tmp/example
$ ls -l /tmp/example
-rwsr-sr-x  1 david  users    0 Apr 19 11:21 /tmp/example
```

If the octal values confuse you, you can use letters with **chmod**. The permission groups are represented as:

Owner	u
Group	g
World	o
All of the above	a

To do the above, we would have to use several command lines:

```
$ chmod a+rx /tmp/example
$ chmod u+w /tmp/example
$ chmod ug+s /tmp/example
```

Some people prefer the letters over the numbers. Either way will result in the same set of permissions.

We mentioned set user ID and set group ID permissions in several places above. You may be wondering what this is. Normally when you run a program, it is operating under your user account. That is, it has all the permissions that you as a user have. The same is true for the group. When you run a program, it executes under your current group. With set user ID permissions, you can force the program to always run as the program owner (such as `root`). Set group ID is the same, but for the group.

Be careful with this, set user ID and set group ID programs can open major security holes on your system. If you frequently set user ID programs that are owned by `root`, you are allowing anyone to run that program and run it as `root`. Since `root` has no restrictions on the system, you can see how this would pose a major security problem. In short, it's not bad to use set user ID and set group ID permissions, just use common sense.

Links

Links are pointers between files. With links, you can have files exist in many locations and be accessible by many names. There are two types of links: hard and soft.

Hard links are names for a particular file. They can only exist within a single directory and are only removed when the real name is removed from the system. These are useful in some cases, but many users find the soft link to be more versatile.

The soft link, also called a symbolic link, can point to a file outside of its directory. It is actually a small file containing the information it needs. You can add and remove soft links without affecting the actual file.

Links do not have their own set of permissions or ownerships, but instead reflect those of the file they point to. Slackware uses mostly soft links. Here is a common example:

```
$ ls -l /bin/sh
lrwxrwxrwx  1 root      root      4 Apr  6 12:34 /bin/sh -> bash
```

The **sh** shell under Slackware is actually **bash**. Removing links is done using **rm**. The **ln** command is used to create links. These commands will be discussed in more depth in [Chapter 10](#).

Mounting Devices

As was previously discussed in [the section called *File System Layout* in Chapter 4](#), all the drives and devices in your computer are one big filesystem. Various hard drive partitions, CD-ROMs, and floppies are all placed in the same tree. In order to attach these drives to the filesystem so that you can access them, you have to use the **mount**(1) and **umount**(1) commands.

Some devices are automatically mounted when you boot up your computer. These are listed in the `/etc/fstab` file. Anything that you want to be mounted automatically gets an entry in that file. For other devices, you'll have to issue a command every time you want to use the device.

fstab

Let's look at an example of the `/etc/fstab` file:

<code>/dev/sda1</code>	<code>/</code>	<code>ext2</code>	<code>defaults</code>	<code>1</code>	<code>1</code>
<code>/dev/sda2</code>	<code>/usr/local</code>	<code>ext2</code>	<code>defaults</code>	<code>1</code>	<code>1</code>
<code>/dev/sda4</code>	<code>/home</code>	<code>ext2</code>	<code>defaults</code>	<code>1</code>	<code>1</code>
<code>/dev/sdb1</code>	<code>swap</code>	<code>swap</code>	<code>defaults</code>	<code>0</code>	<code>0</code>
<code>/dev/sdb3</code>	<code>/export</code>	<code>ext2</code>	<code>defaults</code>	<code>1</code>	<code>1</code>
<code>none</code>	<code>/dev/pts</code>	<code>devpts</code>	<code>gid=5,mode=620</code>	<code>0</code>	<code>0</code>
<code>none</code>	<code>/proc</code>	<code>proc</code>	<code>defaults</code>	<code>0</code>	<code>0</code>
<code>/dev/fd0</code>	<code>/mnt</code>	<code>ext2</code>	<code>defaults</code>	<code>0</code>	<code>0</code>
<code>/dev/cdrom</code>	<code>/cdrom</code>	<code>iso9660</code>	<code>ro</code>	<code>0</code>	<code>0</code>

The first column is the device name. In this case, the devices are five partitions spread out across two SCSI hard drives, two special filesystems that don't need a device, a floppy, and a CD-ROM drive. The second column is where the device will be mounted. This needs to be a directory name, except in the case of a swap partition. The third column is the filesystem type of the device. For normal Linux filesystems, this will be **ext2** (second extended filesystem). CD-ROM drives are **iso9660**, and Windows-based devices will either be **msdos** or **vfat**.

The fourth column is a listing of options that apply to the mounted filesystem. `defaults` is fine for just about everything. However, read-only devices should be given the **ro** flag. There are a lot of options that can be used. Check the `fstab(5)` man page for more information. The last two columns are used by **fsck** and other commands that need to manipulate the devices. Check the man page for that information, as well.

When you install Slackware Linux, the setup program will build much of the `fstab` file. Then only time you will need to edit it is if you add disks or want devices to be automatically mounted at boot time.

mount and umount

Attaching another device to your filesystem is easy. All you have to do is use the **mount** command, along with a few options. Using **mount** can also be made much more simple if the device has an entry in the `/etc/fstab` file. For example, let's say that I wanted to mount my CD-ROM drive and that my `fstab` file looked like the example from the previous section. I would call **mount** like so:

```
# mount /cdrom
```

Since there is an entry in `fstab` for that mount point, **mount** knows what options to use. If there wasn't an entry for that device, I would have to use several options for **mount**:

```
# mount -t iso9660 -o ro /dev/cdrom /cdrom
```

That command line includes the same information as the example `fstab` did, but we'll go over all the parts anyways. The **-t iso9660** is the filesystem type of the device to mount. In this case, it would be the `iso9660` filesystem which is what CD-ROM drives most commonly use. The **-o ro** tells **mount** to mount the device read-only. The **/dev/cdrom** is the name of the device to mount, and **/cdrom** is the location on the filesystem to mount the drive.

Before you can remove a floppy, CD-ROM, or other removable device that is currently mounted, you'll have to unmount it. That is done using the **umount** command. Don't ask where the `n` went because we couldn't tell you. You can use either the mounted device or the mount point as the argument to **umount**. For example, if you wanted to unmount the CD-ROM from the previous example, either of these commands would work:

```
# umount /dev/cdrom
# umount /cdrom
```

[Prev](#)

Links

[Home](#)

[Up](#)

[Next](#)

NFS Mounts

NFS Mounts

NFS stands for the Network Filesystem. It is not really part of the real filesystem, but can be used to add parts to the mounted filesystem.

Large Unix environments often times share the same programs, sets of home directories, and mail spool. The problem of getting the same copy to each machine is solved with NFS. We can use NFS to share one set of home directories between all of the workstations. The workstations then mount that NFS share as if it were on their own machines.

See [the section called *NFS \(Network File System\)* in Chapter 5](#) and the man pages for `exports(5)`, `nfsd(8)`, and `mountd(8)` for more information.

Summary

In this chapter, you should have gained knowledge of ownerships and permissions. You should know why these exist and how to set them. You also should know about links between files, mounting devices, and NFS mounts. These three things are important aspects of the filesystem. You should have a basic idea of how to use them.

Chapter 10. Handling Files and Directories

Table of Contents

[ls](#)[cd](#)[more](#)[less](#)[cat](#)[touch](#)[echo](#)[mkdir](#)[ln](#)[cp](#)[mv](#)[rm](#)[rmdir](#)[Summary](#)

Slackware Linux aims to be the most Unix-like it can be. Traditionally, Unix operating systems have been command-line oriented. We do have a graphical user interface in Slackware, but the command-line is still the main level of control for the system. Therefore, it is important to understand some of the basic file management commands.

The following sections explain the common file management commands and examples of how they are used. There are many other commands, but these will help you get started. Also, the commands are only briefly discussed here. You will find more detail in the accompanying man pages for each command.

ls

This command lists files in a directory. Windows and DOS users will notice its similarity to the **dir** command. By itself, **ls(1)** will list the files in the current directory. To see what's in your root directory, you could issue these commands:

```
$ cd /
$ ls
bin    cdr    dev    home   lost+found  proc  sbin    tmp    var
boot  cdrom  etc    lib    mnt        root  suncd   usr    vmlinuz
```

The problem a lot of people have with that output is that you cannot easily tell what is a directory and what is a file. Some users prefer that **ls** add a type identifier to each listing, like this:

```
$ ls -FC
bin/    cdr/    dev/    home/    lost+found/  proc/    sbin/    tmp/    var/
boot/   cdrom/  etc/    lib/     mnt/         root/    suncd/   usr/    vmlinuz
```

Directories get a slash at the end of the name, executable files get an asterisk at the end of the name, and so on.

ls can also be used to get other statistics on files. For example, to see the creation dates, owners, and permissions, you would look at a long listing:

```
$ ls -l
drwxr-xr-x  2 root    bin          4096 May  7  1994 bin/
drwxr-xr-x  2 root    root         4096 Feb 24 03:55 boot/
drwxr-xr-x  2 root    root         4096 Feb 18 01:10 cdr/
drwxr-xr-x 14 root    root         6144 Oct 23 18:37 cdrom/
drwxr-xr-x  4 root    root        28672 Mar  5 18:01 dev/
drwxr-xr-x 10 root    root         4096 Mar  8 03:32 etc/
drwxr-xr-x  8 root    root         4096 Mar  8 03:31 home/
drwxr-xr-x  3 root    root         4096 Jan 23 21:29 lib/
drwxr-xr-x  2 root    root        16384 Nov  1 08:53 lost+found/
drwxr-xr-x  2 root    root         4096 Oct  6  1997 mnt/
dr-xr-xr-x 62 root    root           0 Mar  4 15:32 proc/
drwxr-x--x 12 root    root         4096 Feb 26 02:06 root/
drwxr-xr-x  2 root    bin          4096 Feb 17 02:02 sbin/
drwxr-xr-x  5 root    root        2048 Oct 25 10:51 suncd/
drwxrwxrwt  4 root    root       487424 Mar  7 20:42 tmp/
drwxr-xr-x 21 root    root         4096 Aug 24  1999 usr/
drwxr-xr-x 18 root    root         4096 Mar  8 03:32 var/
-rw-r--r--  1 root    root       461907 Feb 22 20:04 vmlinuz
```

Suppose you want to get a listing of the hidden files in the current directory. This command will do just that:

```
$ ls -a
.          bin    cdrom  home      mnt    sbin    usr
..         boot   dev    lib        proc   suncd   var
.pwrchute_tmp cdr    etc    lost+found root    tmp     vmlinuz
```

Files beginning with a period (called dot files) are hidden when you run **ls**. You will only see them if you pass the **-a** option.

There are many more options that can be found in the online manual page. Don't forget that you can combine options that you pass to **ls**.

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

cd

cd

The **cd** command is used to change working directories. You simply type **cd** followed the path name to change to. Here's some examples:

```
darkstar:~$ cd /bin
darkstar:/bin$ cd usr
bash: cd: usr: No such file or directory
darkstar:/bin$ cd /usr
darkstar:/usr$
```

Notice that without the preceding slash, it tries to change to a directory in the current directory.

The **cd** command is not like the other commands. It is a builtin shell command. Shell builtins are discussed in [the section called *Environment Variables* in Chapter 8](#). This may not make any sense to you right now. Basically it means there is no man page for this command. Instead, you have to use the shell help. Like this:

```
$ help cd
```

It will display the options for **cd** and how to use them.

more

more(1) is what we call a pager utility. Oftentimes the output of a particular command is too big to fit on one screen. The individual commands do not know how to fit their output to separate screens. They leave this job to the pager utility.

The **more** command breaks the output into individual screens and waits for you to press the space bar before continuing on to the next screen. Pressing the enter key will advance the output one line. Here is a good example:

```
$ cd /usr/bin
$ ls -l
```

That should scroll for a while. To break up the output screen by screen, just pipe it through **more**:

```
$ ls -l | more
```

That is the pipe character (shift backslash). The pipe is short for saying take the output of **ls** and feed it into **more**. You can pipe just about anything through the **more** command, not just **ls**. Piping is also covered in [the section called *Input/Output Redirection and Piping* in Chapter 8](#).

less

The **more** command is quite handy, but oftentimes you will find that you have advanced past the screen you wanted. **more** does not provide a way to go back. The **less(1)** command provides this functionality. It is used in the same way as the **more** command, so the previous examples apply here too. So, **less** is more than **more**.

cat

cat(1) is short for concatenate . It was originally designed to merge text files into one, but can be used for many other purposes.

To merge two or more files into one, you simply list the files after the **cat** command and then redirect the new output to a file. **cat** works with standard input and standard output, so you have to use the shell redirection characters. For example:

```
$ cat file1 file2 file3 > bigfile
```

This command takes the contents of `file1`, `file2`, and `file3` and merges it all together. The new output is sent to standard out.

One can also use **cat** to display files. Many people cat text files through the **more** or **less** commands, like this:

```
$ cat file1 | more
```

That will display the `file1` file and pipe it through the **more** command so that you only get one screen at a time.

Another common use for **cat** is copying files. You can copy any file around with **cat**, like this:

```
$ cat /bin/bash > ~/mybash
```

The `/bin/bash` program is copied to your home directory and named `mybash` .

cat has many uses and the ones discussed here are just a few. Since **cat** makes extensive use of standard input and standard output, it is ideal for use in shell scripts or part of other complex commands.

touch

touch(1) is used to change the timestamp on a file. You can change access timestamps and modification timestamps with this command. If the file specified does not exist, **touch** will create a zero length file with the name specified. To mark a file with the current system time, you would issue this command:

```
$ touch file1
```

There are several options for **touch**, including options to specify which timestamp to modify, the time to use, and many more. The online manual page discusses these in detail.

echo

The **echo**(1) command displays the specified text on the screen. You specify the string to display after the **echo** command. By default **echo** will display the string and print a newline character after it. You can pass the **-n** option to suppress the printing of the newline. The **-e** option will cause **echo** to search for escape characters in the string and execute them.

mkdir

mkdir(1) will create a new directory. You simply specify the directory to create when you run **mkdir**. This example creates the `hejaz` directory in the current directory:

```
$ mkdir hejaz
```

You can also specify a path, like this:

```
$ mkdir /usr/local/hejaz
```

The **-p** option will tell **mkdir** to make any parent directories. The above example will fail if `/usr/local` does not exist. The **-p** option will create `/usr/local` and `/usr/local/hejaz`:

```
$ mkdir -p /usr/local/hejaz
```

In

ln(1) is used to create links between files. These links can be either hard links or soft (symbolic) links. The differences between the two kinds of links were discussed in [the section called *Links* in Chapter 9](#). If you wanted to make a symbolic link to the directory `/var/media/mp3` and place the link in your home directory, you would do this:

```
$ ln -s /var/media/mp3 ~/mp3
```

The **-s** option tells **ln** to make a symbolic link. The next option is the target of the link, and the final option is what to call the link. In this case, it will just make a file called `mp3` in your home directory that points to `/var/media/mp3`. You can call the link itself whatever you want by just changing the last option.

Making a hard link is just as simple. All you have to do is leave off the **-s** option. Making a hard link out of the previous command would be done as follows:

```
$ ln /var/media/mp3 ~/mp3
```

cp

cp(1) copies files. DOS users will notice its similarity to the copy command. There are many options for **cp**, so you should have a look at the man page before using it.

A common use is to use **cp** to copy a file from one location to another. For example:

```
$ cp hejaz /tmp
```

This copies the hejaz file from the current directory to the /tmp directory.

Many users prefer to keep the timestamps preserved, as in this example:

```
$ cp -a hejaz /tmp
```

This ensures that the timestamps are not modified in the copy.

To recursively copy the contents of a directory to another directory, you would issue this command:

```
$ cp -R adirectory /tmp
```

That will copy the adirectory directory to the /tmp directory.

cp has many more options that are discussed in detail in the online manual page.

mv

mv(1) will move files from one location to another. DOS users will notice the similarity to the move command. You name the source and destination when you run **mv**. This example shows a common use of **mv**:

```
# mv myfile /usr/local/share/hejaz
```

mv has several command line options which are documented in detail in the online manual page.

rm

rm(1) removes files and directory trees. DOS users will notice the similarity to both the `del` and `deltree` commands. **rm** can be very dangerous if you do not watch yourself. Unlike DOS or Windows, Linux does not provide a way to undelete files.

To remove a single file, specify its name when you run **rm**:

```
$ rm file1
```

If the file has write permissions removed, you may get a permission denied error message. To force removal of the file no matter what, pass the **-f** option, like this:

```
$ rm -f file1
```

To remove an entire directory, you use the **-r** and **-f** options together. This is a good example of how to delete the entire contents of your hard drive. You really don't want to do this. But here's the command anyway:

```
# rm -rf /
```

Be very careful with **rm**; you can shoot yourself in the foot. There are several command line options, which are discussed in detail in the online manual page.

rmdir

rmdir(1) removes directories from the filesystem. The directory must be empty before it can be removed. The syntax is simply:

```
$ rmdir <directory>
```

This example will remove the `hejaz` subdirectory in the current working directory:

```
$ rmdir hejaz
```

If that directory does not exist, **rmdir** will tell you. You can also specify a full path to a directory to remove, as this example shows:

```
$ rmdir /tmp/hejaz
```

That example will try to remove the `hejaz` directory inside the `/tmp` directory.

You can also remove a directory and all of its parent directories by passing the `-p` option.

```
$ rmdir -p /tmp/hejaz
```

This will first try to remove the `hejaz` directory inside `/tmp`. If that is successful, it will try to remove `/tmp`. **rmdir** will continue this until an error is encountered or the entire tree specified is removed.

Summary

This chapter covered a lot of programs that manipulate files and directories. You should know how to create, delete, and move just about anything on the filesystem. You should also know how to list and touch files if needed. Finally, you should know why **rm -rf /** is a very bad idea.

Chapter 11. Process Control

Table of Contents

[Backgrounding](#)[Foregrounding](#)[ps](#)[kill](#)[top](#)[Summary](#)

Every program that is running is called a process. These processes range from things like the X Window System to system programs (daemons) that are started when the computer boots. Every process runs as a particular user. Processes that are started at boot time usually run as root or nobody. Processes that you start will run as you. Processes started as other users will run as those users.

You have control over all the processes that you start. Additionally, root has control over all processes on the system, including those started by other users. Processes can be controlled and monitored through several programs, as well as some shell commands.

Backgrounding

Programs started from the command line start up in the foreground. This allows you to see all the output of the program and interact with it. However, there are several occasions when you'd like the program to run without taking up your terminal. This is called running the program in the background, and there are a few ways to do it.

The first way to background a process is by adding an ampersand to the command line when you start the program. For example, assume you wanted to use the command line mp3 player **amp** to play a directory full of mp3s, but you needed to do something else on the same terminal. The following command line would start up amp in the background:

```
$ amp *.mp3 &
```

The program will run as normal, and you are returned to a prompt.

The other way to background a process is to do so while it is running. First, start up a program. While it is running, hit **control+z**. This suspends the process. A suspended process is basically paused. It momentarily stops running, but can be started up again at any time. Once you have suspended a process, you are returned to a prompt. You can

background the process by typing:

```
$ bg
```

Now, the suspended process is running in the background.

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

Foregrounding

Foregrounding

If you need to interact with a backgrounded process, you can bring it back into the foreground. If you've only got one backgrounded process, you can bring it back by typing:

```
$ fg
```

If the program is not done running, the program will take control over your terminal and you will not be returned to a prompt. Sometimes, the program will finish running while backgrounded. In this instance, you'll get a message like this:

```
[1]+  Done                  /bin/ls $LS_OPTIONS
```

That tells you that the backgrounded process (in this case **ls** - not terribly interesting) has completed.

It is possible to have several processes backgrounded at once. When this happens, you'll need to know which process you want to bring back to the foreground. Just typing **fg** will foreground the process that was last backgrounded. What if you had a whole list of processes in the background? Luckily, bash includes a command to list all the processes. It's called **jobs** and gives output like so:

```
$ jobs
[1]  Stopped                  vim
[2]- Stopped                  amp
[3]+ Stopped                  man ps
```

This shows you a list of all the processes that are backgrounded. As you can see, they are all stopped. This means that the processes are suspended. The number is a sort of ID for all the backgrounded processes. The ID with a plus sign beside it (man ps) is the process that will be foregrounded if you just type **fg**.

If you wanted to foreground **vim**, you would type:

```
$ fg 1
```

and **vim** would spring back up to the console. Backgrounding processes can be very useful if you only have one terminal open over a dialup connection. You can have several programs running on that one terminal, periodically switching back and forth between them.

ps

So now you know how to switch back and forth between several processes that you've started from the command line. And you also know that there are lots of processes running all the time. So how do you list all of these programs? Well, you make use of the **ps**(1) command. This command has a lot of options, so we'll only cover the most important ones here. For a complete listing, see the man page for **ps**. Man pages are covered in-depth in [the section called *man* in Chapter 2](#).

Simply typing **ps** will get you a listing of the programs running on your terminal. Many times, that will be a very short listing:

```
$ ps
  PID TTY          TIME CMD
 7923 ttyp0      00:00:00 bash
 8059 ttyp0      00:00:00 ps
```

Even though this is not a lot of processes, the information is very typical. You'll get the same columns using regular **ps** no matter how many processes are running. So what does it all mean?

Well, the PID is the process ID. All running processes are given a unique identifier. On 2.2.x kernels, this process ID can be anywhere between 1 and 32767. Each process is assigned the next free PID. When a process quits (or is killed, as you will see in the next section), it gives up its PID. When the max PID is reached, the next free one will wrap back around to the lowest free one. This will most likely change in the upcoming 2.4 kernel series and the introduction of 32-bit PIDs.

The TTY column indicated which terminal the process is running on. Doing a plain **ps** will only list all the programs running on the current terminal, so all the processes give the same information in the TTY column. As you can see, both processes listed are running on ttyp0. This indicates that they are either running remotely or from an X terminal of some variety.

The TIME column indicated how much CPU time the process has been running. This is different from the actual amount of time that a process runs. Remember that Linux is a multitasking operating system. There are many processes running all the time, and these processes each get a small portion of the processor's time. So, the TIME column should show much less time for each process than it actually takes to run. If you see more than several minutes in the TIME column, it could mean that something is wrong.

Finally, the CMD column shows what the program actually is. It only lists the base name of the program, not any command line options or similar information. To get that information, you'll need to use one of the many options to **ps**. We'll discuss that shortly.

You can get a complete listing of the processes running on your system using the right combination of options. This will probably result in a long listing of processes (fifty-five on my laptop as I write this sentence), so I'll abbreviate the output:

```
$ ps -ax
  PID TTY          STAT TIME  COMMAND
    1 ?            S     0:03   init [3]
    2 ?            SW     0:13 [kflushd]
    3 ?            SW     0:14 [kupdate]
    4 ?            SW     0:00 [kpiod]
    5 ?            SW     0:17 [kswapd]
   11 ?            S     0:00 /sbin/kerneld
   30 ?            SW     0:01 [cardmgr]
   50 ?            S     0:00 /sbin/rpc.portmap
   54 ?            S     0:00 /usr/sbin/syslogd
   57 ?            S     0:00 /usr/sbin/klogd -c 3
   59 ?            S     0:00 /usr/sbin/inetd
   61 ?            S     0:04 /usr/local/sbin/sshd
   63 ?            S     0:00 /usr/sbin/rpc.mountd
```

```

65 ?      S      0:00 /usr/sbin/rpc.nfsd
67 ?      S      0:00 /usr/sbin/crond -l10
69 ?      S      0:00 /usr/sbin/atd -b 15 -l 1
77 ?      S      0:00 /usr/sbin/apmd
79 ?      S      0:01 gpm -m /dev/mouse -t ps2
94 ?      S      0:00 /usr/sbin/automount /auto file /etc/auto.misc
106 tty1   S      0:08 -bash
108 tty3   SW     0:00 [agetty]
109 tty4   SW     0:00 [agetty]
110 tty5   SW     0:00 [agetty]
111 tty6   SW     0:00 [agetty]
[output cut]

```

Most of these processes are started at boot time on most systems. I've made a few modifications to my system, so your mileage will most likely vary. However, you will see most of these processes on your system too. As you can see, these options display command line options to the running processes. It also brings up a few more columns and some other interesting output.

First, you'll notice that most of these processes are listed as running on `tty ?`. Those are processes that were started from a no-longer active terminal. Therefore, they are no longer attached to any particular terminal.

Second, there is a new column: `STAT`. It shows the status of the process. `S` stands for sleeping: the process is waiting for something to happen. `Z` stands for a zombied process. A zombied processes is one whose parent has died, leaving the child processes behind. This is not a good thing.

If you want to see even more information about the running processes, try this out:

```

$ ps -aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root             1  0.0  0.0    344    80 ?        S      Mar02    0:03 init [3]
root             2  0.0  0.0      0      0 ?        SW     Mar02    0:13 [kflushd]
root             3  0.0  0.0      0      0 ?        SW     Mar02    0:14 [kupdate]
root             4  0.0  0.0      0      0 ?        SW     Mar02    0:00 [kpiod]
root             5  0.0  0.0      0      0 ?        SW     Mar02    0:17 [kswapd]
root            11  0.0  0.0   1044    44 ?        S      Mar02    0:00 /sbin/kerneld
root            30  0.0  0.0   1160      0 ?        SW     Mar02    0:01 [cardmgr]
bin             50  0.0  0.0   1076   120 ?        S      Mar02    0:00 /sbin/rpc.port
root            54  0.0  0.1   1360   192 ?        S      Mar02    0:00 /usr/sbin/sysl
root            57  0.0  0.1   1276   152 ?        S      Mar02    0:00 /usr/sbin/klog
root            59  0.0  0.0   1332    60 ?        S      Mar02    0:00 /usr/sbin/inet
root            61  0.0  0.2   1540   312 ?        S      Mar02    0:04 /usr/local/sbi
root            63  0.0  0.0   1796    72 ?        S      Mar02    0:00 /usr/sbin/rpc.
root            65  0.0  0.0   1812    68 ?        S      Mar02    0:00 /usr/sbin/rpc.
root            67  0.0  0.2   1172   260 ?        S      Mar02    0:00 /usr/sbin/cron
root            77  0.0  0.2   1048   316 ?        S      Mar02    0:00 /usr/sbin/apmd
root            79  0.0  0.1   1100   152 ?        S      Mar02    0:01 gpm
root            94  0.0  0.2   1396   280 ?        S      Mar02    0:00 /usr/sbin/auto
chris          106  0.0  0.5   1820   680 tty1     S      Mar02    0:08 -bash
root           108  0.0  0.0   1048      0 tty3     SW     Mar02    0:00 [agetty]
root           109  0.0  0.0   1048      0 tty4     SW     Mar02    0:00 [agetty]
root           110  0.0  0.0   1048      0 tty5     SW     Mar02    0:00 [agetty]
root           111  0.0  0.0   1048      0 tty6     SW     Mar02    0:00 [agetty]
[output cut]

```

That's a whole lot of information. Basically, it adds information including what user started the process, how much of the system resources the process is using (the `%CPU`, `%MEM`, `VSZ`, and `RSS` columns), and on what date the process was started. Obviously, that's a lot of information that could come in handy for a system administrator. It also brings up another point: the information now goes off the edge of the screen so that you cannot see it all. Well, the `-w` option will do that.

It's not terribly pretty, but it does the job. You've now got the complete listings for each process. There's even more information that you can display about each process. Check out the very in-depth man page for `ps`. However, the options shown are the most popular ones and will be the ones you need to use the most often.

[Prev](#)

Foregrounding

[Home](#)

[Up](#)

[Next](#)

kill

kill

On occasion, programs misbehave and you'll need to put them back in line. The program for this kind of administration is called **kill**(1), and it can be used for manipulating processes in several ways. The most obvious use of **kill** is to kill off a process. You'll need to do this if a program has run away and is using up lots of system resources, or if you're just sick of it running.

In order to kill off a process, you'll need to know its PID or its name. To get the PID, use the **ps** command as was discussed in the last section. For example, to kill off process 4747, you'd issue the following:

```
$ kill 4747
```

Note that you'll have to be the owner of the process in order to kill it. This is a security feature. If you were allowed to kill off processes started by other users, it would be possible to do all sorts of malicious things. Of course, root can kill off any process on the system.

There's another variety of the kill command called **killall**(1). This program does exactly what it says: it kills all the running processes that have a certain name. If you wanted to kill off all the running **vim** processes, you could type the following command:

```
$ killall vim
```

Any and all **vim** processes you have running will die off. Doing this as root would kill off all the **vim** processes running for all users. This brings up an interesting way to kick everyone (including yourself) off the system:

```
# killall bash
```

Sometimes a regular **kill** doesn't get the job done. Certain processes will not die with a **kill**. You'll need to use a more potent form. If that pesky PID 4747 wasn't responding to your kill request, you could do the following:

```
$ kill -9 4747
```

That will almost certainly cause process 4747 to die. You can do the same thing with **killall**. What this is doing is sending a different signal to the process. A regular kill sends a SIGTERM (terminate) signal to the process. **kill -9** sends a SIGKILL (kill) signal to the process. There's a whole list of signals at your disposal. You can get a listing of signals by typing the following:

```
$ kill -l
```

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGBUS	8) SIGFPE
9) SIGKILL	10) SIGUSR1	11) SIGSEGV	12) SIGUSR2
13) SIGPIPE	14) SIGALRM	15) SIGTERM	17) SIGCHLD
18) SIGCONT	19) SIGSTOP	20) SIGTSTP	21) SIGTTIN
22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO
30) SIGPWR			

The number must be used for **kill**, while the name minus the leading **SIG** can be used with **killall**. Here's another example:

```
$ killall -KILL vim
```

A final use of **kill** is to restart a process. Sending a SIGHUP will cause most processes to re-read their configuration files. This is especially helpful for telling system processes to re-read their config files after editing.

[Prev](#)

ps

[Home](#)

[Up](#)

[Next](#)

top

top

Finally, there's a command you can use to display updating information about the processes running on the system. This command is called **top**(1), and is started like so:

```
$ top
```

This will display a full screen of information about the processes running on the system, as well as some overall information about the system. This includes load average, number of processes, the CPU status, free memory information, and details about processes including PID, user, priority, CPU and memory usage information, running time, and program name.

Figure 11-1. Example output of the top program.

```

6:47pm up 1 day, 18:01, 1 user, load average: 0.02, 0.07, 0.02
61 processes: 59 sleeping, 2 running, 0 zombie, 0 stopped
CPU states: 2.8% user, 3.1% system, 0.0% nice, 93.9% idle
Mem: 257992K av, 249672K used, 8320K free, 51628K shrd, 78248K buff
Swap: 32764K av, 136K used, 32628K free, 82600K cached

```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	LIB	%CPU	%MEM	TIME	COMMAND
112	root	12	0	19376	18M	2468	R	0	3.7	7.5	55:53	X
4947	david	15	0	2136	2136	1748	S	0	2.3	0.8	0:00	screenshot
3398	david	7	0	20544	20M	3000	S	0	1.5	7.9	0:14	gimp
4946	root	12	0	1040	1040	836	R	0	1.5	0.4	0:00	top
121	david	4	0	796	796	644	S	0	1.1	0.3	25:37	wmSMPmon
115	david	3	0	2180	2180	1452	S	0	0.3	0.8	1:35	wmaker
4948	david	16	0	776	776	648	S	0	0.3	0.3	0:00	xwd
1	root	1	0	176	176	148	S	0	0.1	0.0	0:13	init
189	david	1	0	6256	6256	4352	S	0	0.1	2.4	3:16	licq
4734	david	0	0	1164	1164	916	S	0	0.1	0.4	0:00	rxvt
2	root	0	0	0	0	0	SW	0	0.0	0.0	0:08	kflushd
3	root	0	0	0	0	0	SW	0	0.0	0.0	0:06	kupdate
4	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kpiod
5	root	0	0	0	0	0	SW	0	0.0	0.0	0:04	kswapd
31	root	0	0	340	328	284	S	0	0.0	0.1	0:00	kerneld
51	root	0	0	48	48	32	S	0	0.0	0.0	0:00	dhcpcd
53	bin	0	0	316	304	236	S	0	0.0	0.1	0:00	rpc.portmap
57	root	0	0	588	588	488	S	0	0.0	0.2	0:01	syslogd

It's called **top** because the most CPU intensive programs will be listed at the top. An interesting note is that **top** will be listed first on most inactive (and some active) systems because of its CPU utilization. However, **top** is quite useful for determining what program is misbehaving and needs to be killed off.

Summary

This chapter discussed what a process is and how you can control one. This includes backgrounding and foregrounding, as well as using **ps**, **top**, and **kill** to keep them in line. You should be able to determine what processes are running on your system and know how to get rid of them if they stop behaving.

Chapter 12. Essential System Administration

Table of Contents

[Users and Groups](#)

[Shutting Down Properly](#)

[Summary](#)

You are the administrator of any computers that you have root on. This might be your desktop box with one or two users, or it might be a big server with several hundred. Regardless, you'll need to know how to manage users and safely bring down the system. They are both seemingly simple, but do have some quirks to get used to. In addition, you'll have to deal with some of the ideas behind how the password system works.

Users and Groups

Supplied Scripts

The easiest way to manage users and groups is with the supplied scripts and programs. Slackware includes the programs **adduser**, **userdel**(8), **chfn**(1), **chsh**(1), and **passwd**(1) for dealing with users. Slackware includes **groupadd**(8), **groupdel**(8), and **groupmod**(8) for dealing with groups. With the exception of **chfn**, **chsh**, and **passwd**, these are programs that can only be run as root and are therefore located in `/usr/sbin`. **chfn**, **chsh**, and **passwd** can be run by anyone and are located in `/usr/bin`.

Users are added with the **adduser** program. We'll start out by going through the whole procedure, showing all the questions that are asked and a brief description of what everything means. The default answer is in the brackets, and can be chosen for almost all the questions, unless you really want to change something.

```
# adduser
Login name for new user (8 characters or less) []: jellyd
```

This is the name that the user will use to login. It needs to be eight characters or less, because all the login utilities expect it to be so. Generally, you should only use lowercase characters unless you want to type uppercase letters in inconvenient places.

```
User id for jellyd [ defaults to next available ]:
```

The user ID (UID) is how ownerships are really determined in Linux. Each user has a unique number, starting at 1000 in Slackware. You can pick a UID for the new user, or you can just let **adduser** assign the user the next free one.

```
Initial group for jellyd [users]:
```

All users are placed into the `users` group by default. You might want to place the new user into a different group, but it is not recommended.

```
Additional groups for jellyd (seperated with commas, no spaces) []:
```

This question allows you to place the new user into additional groups. It is possible for a user to be in several groups at the same time. This is useful if you have established groups for things like modifying web site files, playing games, and so on.

```
jellyd's home directory [/home/jellyd]:
```

Home directories default to being placed under /home. If you run a very large system, it's possible that you have moved the home directories to a different scheme. This allows you to change where the user's home directory will be. You can also disable an account by changing a person's home directory to something like /bin/false, though this is not the recommended method.

```
jellyd's shell [/bin/bash]:
```

bash is the default shell for Slackware Linux, and will be fine for most people. If your new user comes from a Unix background, they may be familiar with a different shell. You can change their shell now, or they can change it themselves later using the **chsh** command.

```
jellyd's account expiry date (YYYY-MM-DD) []:
```

Accounts can be set up to expire on a specified date. By default, there is no expiration date. You can change that, if you'd like. This option might be useful for people running an ISP who might want to make an account expire upon a certain date, unless they receive the next year's payment.

```
OK, I'm about to make a new account. Here's
what you entered so far:
```

```
New login name: jellyd
New UID: [Next available]
Initial group: users
Additional groups: [none]
Home directory: /home/jellyd
Shell: /bin/bash
Expiry date: [no expiration]
```

```
This is it... if you want to bail out, hit Control-C.
Otherwise, press ENTER to go ahead and make the account.
```

You now see all the information that has been entered about the new account, and are given the opportunity to stop. If you entered something incorrectly, you'll have to hit Control-C and start over. Otherwise, you can hit enter and the account will be made.

```
Making new account...
```

```
Changing the user information for jellyd
Enter the new value, or press return for the default
  Full Name []: Jeremy
  Room Number []: Smith 130
  Work Phone []:
  Home Phone []:
  Other:
```

All of this information is optional. You don't have to enter any of this if you don't want to, and the user can change it at any time using **chfn**. However, it might be useful to enter at least the full name and phone numbers, just in case you need to get in touch with the person.

```
Changing password for jellyd
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
Re-enter new password:
Password changed.
Done...
```

You'll have to enter a password for the new user. Generally, if the new user is not physically present at this point, you'll just pick some default password and tell the user to change it to something more secure.



Choosing a Password

Having a secure password is the first line of defense against getting cracked. You do not want to have an easily guessed password, because that makes it easier for someone to break into your system. Ideally, a secure password would be a random string of characters, including upper and lowercase letters, numbers, and random characters. Just remember that a tab character might not be a wise choice, depending on what kinds of computers you'll be logging in from.

In general, just use common sense: don't pick a password that is someone's birthday, a common phrase, something found on your desk, or anything that is easily associated with you. `secure1` is also bad.

Removing users is not difficult at all. Just run **userdel** with the name of the account to remove. You'll have to make sure that the user is not logged in, and that no processes are running as that user. Also, remember that once you've deleted the user, they're gone.

```
# userdel jellyd
```

Doing this would remove that annoying `jellyd` guy from your system. Good riddance :) This removes the user from the `/etc/passwd` and `/etc/group` files, but doesn't remove the user's home directory. If you wanted to remove the home directory as well, you would do the following:

```
# userdel -r jellyd
```

Temporarily disabling an account will be covered in [the section called *Changing Passwords*](#) since that involves modifying the user's password. Changing account information is covered in [the section called *Changing Passwords*](#) and [the section called *Changing User Information*](#).

The programs to add and remove groups are very simple. **groupadd** will just add another entry to the `/etc/group` file with a unique group ID, while **groupdel** will remove the specified group. It is up to you to go in and edit `/etc/group` to add users to a specific group.

You create a group like so:

```
# groupadd cvs
```

And remove it like so:

```
# groupdel cvs
```

By hand

Of course, it is possible to add, modify, and remove users and groups by hand. After looking through this procedure, you'll probably find it much more convenient to use the scripts, though.

First, we'll add a new user to the `/etc/passwd(5)`, `/etc/shadow(5)`, and `/etc/group(5)` files. The

`passwd` file holds some information about the user, but (strangely enough) not their password. The `passwd` file has to be readable by anyone, but you don't want encrypted passwords world-readable because this gives would-be crackers a good place to start. So the encrypted passwords are kept in the `shadow` file, which is only readable by root, and everyone's password is entered into the `passwd` file as `x`. The group file lists all the groups and who is in each.

Let's go ahead and examine the `/etc/passwd` file and figure out how to add someone. A typical entry in `passwd` looks like this:

```
chris:x:1000:100:Chris Lumens,Room 2,,:/home/chris:/bin/bash
```

Each line is an entry for one person, and fields on each line are separated by a colon. The fields are the login name, encrypted password (`x` for everyone on a Slackware system, since we use the shadow password suite), user ID, group ID, the optional finger information separated by commas, home directory, and shell. What you have to do in this file is add a new line onto the end, filling in the appropriate information.

Make sure that the password is an **x**, that the user ID is unique, that they are in group 100 (the `users` group under Slackware), and that they have a valid shell.

Next, we'll need to add an entry in the `/etc/shadow` file, which holds the passwords. A typical entry looks like this:

```
chris:$1$w9bsw/N9$UWLr2bRER6YyBS.CAEp7R.:11055:0:99999:7:::
```

Again, each line is an entry for one person and the fields are separated by colons. The fields are the login name, encrypted password, days since the Epoch (January 1, 1970) that the password was last changed, days before the password may be changed, days after which the password must be changed, days before password expiration that the user is notified, days after expiration that the account is disabled, days since the Epoch that the account is disabled, and a reserved field.

As you can see, most of that is for account expiration information. If you aren't using expiration information, you only need to fill in a few fields with some special values. Otherwise, you'll need to do some calculations and decision making before you can fill those fields in. For our new user, put some random garbage in the password field. Don't worry about what the password is right now because you're going to change it in a minute. The only character you cannot include in the password field is a colon. Leave the `days since password was changed` field blank as well. Fill in **0**, **99999**, and **7** just as you see in the example entry, and leave the other fields blank.

For those of you who see my encrypted password above and think you've got a leg up on breaking into my system, go right ahead. If you can crack that password, you'll know the password to a firewalled test system. Now that's useful :)

Since everyone is a member of the `users` group by default, you won't need to add the new user to it. If you want to create a new group or add the new user to other groups, you'll need to modify the `/etc/group` file. Here is a typical entry:

```
cvs::102:chris,logan,david,root
```

The fields are group name, group password, group ID, and group members. Creating a new group is a simple matter of adding a new line with a unique group ID and listing all the people you want to be in the group. Any users that are in this new group and are logged in will have to log out and log back in for those changes to take effect.

Now, go back and use the **passwd** command to create a new password for the user. Then, use **mkdir** to create the new user's home directory in the location you entered into the `/etc/passwd` file.

If you've installed **sendmail**(8) on your system and actively use mail, you will need to create a new file in `/var/spool/mail` with the proper permissions and ownerships for this new user. Here's an example:

```
# touch /var/spool/mail/jellyd
# chown jellyd.users /var/spool/mail/jellyd
# chmod 660 /var/spool/mail/jellyd
```

Those commands would create a mail file for the new user `jellyd` and set up the correct ownerships and permissions.

Removing a user is a simple matter of getting rid of everything you just created. Remove the user's entry from `/etc/passwd` and `/etc/group`. Remove their login name from any groups in the `/etc/group` file, remove their mail spool file if they have one, and delete their home directory if needed.

Removing groups is a simple matter of removing the group's entry from `/etc/group`.

Changing Passwords

The **passwd** program changes passwords by modifying the `/etc/shadow` file. This file holds all the passwords for the system in an encrypted format. In order to change your password, you would type:

```
$ passwd
Changing password for chris
Old password:
Enter the new password (minumum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
```

As you can see, you are prompted to enter your old password. It won't appear on the screen as you type it, just like when you log in. Then, you are prompted to enter the new password. **passwd** performs a lot of checks on your new password, and it will complain if your new password doesn't pass its checks. You can ignore its warnings if you want. You will eventually be prompted to enter your new password again for confirmation.

If you are root, you can also change another user's password:

```
# passwd ted
```

You will then have to go through the same procedure as above, except that you won't have to enter the old password. (One of the many benefits of being root...)

If you've got some troublemakers on your system, you can also temporarily disable their accounts. Later on, you can reenable their accounts. Both disabling an account and reenabling an account can be done with **passwd**. To disable an account, do the following as root:

```
# passwd -l david
```

This will change david's password to something that can never match any encrypted value. Then, you would change their password back later by typing this:

```
# passwd -u david
```

Now, david's account is back to normal. Disabling an account might be useful if the user doesn't play by the rules you've set up on your system or if they've exported a very large copy of **xeyes(1)** to your X desktop.

Changing User Information

There are two pieces of information that a user can change about their account at any time: their shell and their finger information. Slackware Linux uses **chsh** (change shell) and **chfn** (change finger) to modify these values.

A user can pick any shell that is listed in the `/etc/shells` file. For most people, **bash** will do just fine.

Others might be familiar with a shell found on their Unix system at work or school and want to use what they already know. The shell is changed using **chsh**:

```
$ chsh
Password:
Changing the login shell for chris
Enter the new value, or press return for the default
    Login Shell [/bin/bash]:
```

After entering your password, enter the full path to the new shell. Make sure that it's listed in the `/etc/shells(5)` file first. root can also change a user's shell by running **chsh** with a username as the argument.

The finger information is the optional information such as your full name, phone numbers, and room number. This can be changed using **chfn**, and follows the same procedure as it did during account creation. As usual, root can change anyone's finger information.

[Prev](#)[Summary](#)[Home](#)[Up](#)[Next](#)[Shutting Down Properly](#)

Shutting Down Properly

It is very important that the system is shut down properly. Simply turning the power off can cause serious filesystem damage. While the system is on, files are in use even if you aren't doing anything. Remember that there are many processes running in the background all the time. These processes are controlling the system and keep a lot of files open. When the system just gets powered-off, these files are not closed properly and become damaged. Depending on what files become damaged, the system might be permanently damaged. In any case, you'll have to go through a long filesystem check procedure on the next reboot.

So, when you go to reboot or power down your computer, it is important to do so the right way. There are several ways of doing this; you can pick whichever one you think is the most fun. Most of the ways for turning off the system can also be applied to rebooting.

The first method is through the **shutdown**(8) program, and it is probably the most popular. **shutdown** can be used to reboot or turn off the system at a given time, and can display a message to all the users of the system telling them that the system is going down.

Basic usage of **shutdown** to turn off the computer is:

```
# shutdown -h now
```

In this case, we are not going to send any special message to the users; they will see **shutdown**'s default message. `now` is the time that we want to shutdown, and the `-h` means to halt the system. This is not a very friendly way to run a multi-user system, but it works just fine on your home computer. A better way on a multiuser system would be to give everyone a little advance warning:

```
# shutdown -h +60
```

This would shutdown the system in one hour (60 minutes), which would be just fine on a normal multiuser system. Really important systems should schedule their downtime far in advance and post warnings about it in the `/etc/motd`(5).

Rebooting the system uses the same command, but substitutes `-r` for `-h`:

```
# shutdown -r now
```

You can use same time notation with **shutdown -r** that you could with **shutdown -h**. There are a lot of other things that you can do with **shutdown** to control when to halt or reboot the machine. See the man page for more details.

The second way of shutting down or powering off the computer is to use the **halt**(8) and **reboot**(8) commands. As the names indicate, **halt** will immediately halt the operating system, and **reboot** will reboot the system. **reboot** is simply a symbolic link to **halt**. They are invoked like so:


```
# halt
# reboot
```

A lower-level way to reboot or shutdown the system is to directly talk to **init**. All the other methods are simply convenient ways to talk to **init**, but you can directly tell it what to do using **telinit**(8) (note that it only has one `l`). Using **telinit** will tell **init** what runlevel to drop into, which will cause a special script to get run. This script will kill or spawn processes as needed for that runlevel. This works for rebooting and shutting down because both of those are special runlevels.

```
# telinit 0
```

Runlevel 0 is halt mode. Telling init to enter runlevel 0 will cause all processes to be killed off, the filesystems unmounted, and the machine to be halted. This is a perfectly acceptable way to bring down the system. On many laptops, this will also cause the machine to be turned off.

```
# telinit 6
```

Runlevel 6 is reboot mode. All processes will be killed off, the filesystems will be unmounted, and the machine will be rebooted. This is a perfectly acceptable method of rebooting the system.

There is one last method of rebooting the system. All the other methods required you to be root. However, it is possible to reboot the machine if you aren't root, provided that you have physical access to the keyboard. Doing a three-fingered salute (control-alt-delete) will cause the machine to be immediately rebooted. What this really does is run the program `/usr/sbin/ctrlaltdel`(8). So, if that program has strange permissions or is not present, pressing all those keys down won't do any good.

[Prev](#)

Essential System
Administration

[Home](#)[Up](#)[Next](#)

Summary

Summary

This chapter discussed the procedures for adding and removing user and groups. You should be able to do these tasks using the provided scripts and by hand, should the need arise. Additionally, you should know what the various parts of adding a user do, techniques for selecting a password, and how to change user information. Finally, you should know how to shut down your computer properly and why it is important to do so. These are all important parts of administering a system whether it is your home computer or a large network server.

Chapter 13. Basic Network Commands

Table of Contents

[ping](#)[finger](#)[telnet](#)[FTP Clients](#)[email](#)[lynx](#)[wget](#)[traceroute](#)[Talking to Other People](#)[Summary](#)

A network consists of several computers connected together. The network can be as simple as a few computers connected in your home or office, or as complicated as a large university network or even the entire Internet. When your computer is part of a network, you have access to those systems either directly or through services like mail and the web.

There are a variety of networking programs that you can use. Some are handy for performing diagnostics to see if everything is working properly. Others (like mail readers and web browsers) are useful for getting your work done and staying in contact with other people.

ping

ping(8) sends an ICMP ECHO_REQUEST packet to the specified host. If the host responds, you get an ICMP packet back. Sound strange? Well, you can `ping` an IP address to see if a machine is alive. If there is no response, you know something is wrong. Here is an example conversation between two Linux users:

User A: Loki's down again.

User B: Are you sure?

User A: Yeah, I tried pinging it, but there's no response.

It's instances like these that make **ping** a very useful day-to-day command. It provides a very quick way to see if a machine is up and connected to the network. The basic syntax is:

```
$ ping <ip address or hostname>
```

There are, of course, several options that can be specified. Check the **ping(1)** man page for more information.

[Prev](#)[Summary](#)[Home](#)[Up](#)[Next](#)[finger](#)

finger

finger(1) will retrieve information about the specified user. You give **finger** a username or an email address and it will try to contact the necessary server and retrieve the username, office, telephone number, and other pieces of information. Here is an example:

```
$ finger johnc@idsoftware.com
```

finger can return the username, mail status, phone numbers, and files referred to as `dot plan` and `dot project`. Of course, the information returned varies with each **finger** server. The one included with Slackware returns the following information by default:

- Username
- Room number
- Home phone number
- Work phone number
- Login status
- Email status
- Contents of the `.plan` file in the user's home directory
- Contents of the `.project` file in the user's home directory

The first four items can be set with the **chfn** command. It stores those values in the `/etc/passwd` file. To change the information in your `.plan` or `.project` file, just edit them with your favorite text editor. They must reside in your home directory and must be called `.plan` and `.project`.

Many users **finger** their own account from a remote machine to quickly see if they have new email. Or, you can see a user's plan or current project. John Carmack of id Software regularly updates his plan file to tell the user community what he is currently working on.

Like many commands, **finger** has options. Check the man page for more information on what special options you can use.

telnet

Someone once stated that **telnet**(1) was the coolest thing he had ever seen on computers. The ability to remotely log in and do stuff on another computer is what separates Unix and Unix-like operating systems from other operating systems.

telnet allows you to log in to a computer, just as if you were sitting at the terminal. Once your username and password are verified, you are given a shell prompt. From here, you can do anything requiring a text console. Compose email, read newsgroups, move files around, and so on. If you are running X and you **telnet** to a machine from an **xterm**, you can run X programs on the remote computer and display them on yours. See [the section called Exporting displays in Chapter 6](#) for more information.

To login to a remote machine, use this syntax:

```
$ telnet <hostname>
```

If the host responds, you will receive a login prompt. Give it your username and password. That's it. You are now at a shell. To quit your **telnet** session, use either the **exit** command or the **logout** command.



IMPORTANT NOTE: **telnet** does not encrypt the information it sends. Everything is sent in plain text, even passwords. It is not advisable to use **telnet** over the Internet. Instead, consider the Secure Shell. It encrypts all traffic and is available for free. See <http://www.ssh.org/> for more information.

FTP Clients

FTP stands for the File Transfer Protocol. It allows you to send and receive files between two computers. There is the FTP server and the FTP client. We discuss the client in this section.

For the curious, the `client` is you. The `server` is the computer that answers your FTP request and lets you login. You will download files from and upload files to the server. The client cannot accept FTP connections, it can only connect to servers.

ftp

To connect to an FTP server, simply run the **ftp**(1) command and specify the host:

```
$ ftp <hostname>
```

If the host is running an FTP server, it will ask for a username and password. You can log in as yourself or as `anonymous`. Anonymous FTP sites are very popular for software archives. For example, to get Slackware Linux via FTP, you must use anonymous FTP.

Once connected, you will be at the `ftp>` prompt. There are special commands for FTP, but they are similar to other standard commands. The following shows some of the basic commands and what they do:

Table 13-1. ftp Commands

Command	Purpose
<code>ls</code>	List files
<code>cd <dirname></code>	Change directory
<code>put <filename></code>	Download a file
<code>put <filename></code>	Upload a file
<code>hash</code>	Toggle hash mark stats indicator
<code>prom</code>	Toggle interactive mode for downloads
<code>mget <mask></code>	Download a file or group of files; wildcard are allowed
<code>mput <mask></code>	Upload a file or group of files; wildcard are allowed
<code>quit</code>	Log off the FTP server

FTP is a fairly simple program to use, but lacks the user interface that many of us are used to nowadays. The man page discusses some of the command line options for **ftp**(1).

ncftp

ncftp(1) (pronounced Nik-F-T-P) is an alternative to the traditional **ftp** client that comes with Slackware. It is still a text-based program, but offers many advantages over **ftp**, including:

- Tab completion

- Bookmarks file
- Passive and non-passive FTP transfer modes
- More liberal wildcard uses
- Command history

By default, **ncftp** will try to log in anonymously to the server you specify. You can force **ncftp** to present a login prompt with the **-u** option. Once logged in, you can use the same commands as in **ftp**, only you'll notice a nicer interface, one that works more like **bash**.

Figure 13-1. Example NcFTP screen.

```
> cd slackware-7.0

> ls
BOOTING.TXT      FILELIST.TXT      UPGRADE.TXT      iso/              slaktest/
COPYING          GLIBC WARNING     bigslack/         kernels/          slakware/
COPYRIGHT.TXT    LOWMEM.TXT        bootdsk5.12/     live/             source/
ChangeLog.txt     MIRRORS.TXT       bootdsk5.144/    modules/          zipslack/
ERRATA           PACKAGES.TXT      contrib/         patches/
FAQ.TXT          README7.TXT       docs/            rootdsk5/

> ls slakware
CHECKSUMS         a13/              a9/              makeflopp*        n9/
CHECKSUMS.md5     a14/              ap1/             n1/               t1/
FILE_LIST         a2/              d1/             n2/              tc11/
MANIFEST.gz       a3/              des1/           n3/              x1/
README           a4/              e1/            n4/              xap1/
a1/              a5/              f1/            n5/              xd1/
a10/             a6/              gtk1/          n6/              xv1/
a11/             a7/              k1/            n7/              y1/
a12/             a8/              kde1/          n8/

ftp.slackware.com      /.1/slackware/slackware-7.0
slackware>
```


email

Electronic mail is one of the most popular things one can do on the Internet. In 1998, it was reported that more electronic mail was sent than regular mail. It is indeed common and useful.

Under Slackware, we provide a standard mail server, and several mail clients. All of the clients discussed are text-based. A lot of Windows users may be against this, but you will find that a text based client is very convenient, especially when checking mail remotely.

pine

pine(1) is not **elm**. Or so the saying goes. The University of Washington created their program for Internet news and email out of a need for an easy mail reader for their students. **pine** is one of the most popular email clients in use today and is available for nearly every flavor of Unix and even Windows.

Figure 13-2. The Pine main menu.

```

PINE 4.21  MAIN MENU                                     Folder: INBOX  No Messages

?      HELP          -  Get help using Pine
C      COMPOSE MESSAGE -  Compose and send a message
I      MESSAGE INDEX  -  View messages in current folder
L      FOLDER LIST    -  Select a folder to view
A      ADDRESS BOOK   -  Update address book
S      SETUP          -  Configure Pine Options
Q      QUIT           -  Leave the Pine program

Copyright 1989-1999. PINE is a trademark of the University of Washington.
[Folder "INBOX" opened with 0 messages]
? Help      F PrevCmd      R RelNotes
C OTHER CMDS [ListFldrs] N NextCmd    K KBlock

```

You will see a menu of commands and a row of command keys at the bottom. **pine** is indeed a complex program, so we will not discuss every feature about it here.

To see what's in your inbox, type **i**. Your messages are listed with their date, author, and subject. Highlight the message you want and press **enter** to view it. Pressing **r** will start a reply to the message. Once you have written the response, type **Ctrl+X** to send it. You can press **i** to get back

to the message listing.

If you want to delete a message, press **d**. It will mark the highlighted message for deletion. **pine** deletes the mail when you exit the program. **pine** also lets you store your mail in folders. You can get a listing of folders by pressing **f**. At the message listing, press **s** to save it to another folder. It will ask for the folder name to write the message to.

pine offers many, many features; you should definitely have a look at the man page for more information. It will contain the latest information about the program.

elm

elm(1) is another popular text-based email client. Though not quite as user friendly as **pine**, it's definitely been around a lot longer.

Figure 13-3. Elm main screen.



```
Mailbox is '/var/spool/mail/root' with 0 messages [ELM 2.5 PL3]

You can use any of the following commands by pressing the first character;
d)delete or u)ndelete mail, m)ail a message, r)eply or f)orward mail, q)uit
To read a message, press <return>. j = move down, k = move up, ? = help

Command: █
```

By default, you are placed in your inbox. The messages are listed with the message number, date, sender, and subject. Use the arrow keys to highlight the message you want. Press Enter to read the message.

To compose a new message, type **m** at the main screen. The **d** key will flag a message for deletion. And the **r** key will reply to the current message you are reading. All of these keys are displayed at the bottom of the screen with a prompt.

The man page discusses **elm** in more detail, so you will probably want to consult that before using **elm**.

mailx

mailx(1) is a command line driven mail client. It is very primitive and offers pretty much nothing in the way of user interfaces. However, **mailx** is handy for times when you need to quickly mail something, scripting a bulk mailer, or something similar.

The basic command line is:

```
$ mailx -s <subject> <to-addr>
```

mailx reads the message body from standard input. So you can cat a file into this command to mail it, or you can just type text and hit **ctrl+D** when finished with the message.

Here is an example of mailing a program source file to another person.

```
$ cat randomfunc.c | mailx -s "Here's that function" \  
asdf@example.net
```

The man page explains more of what **mailx** can do, so you will probably want to have a look at that before using it.

[Prev](#)

FTP Clients

[Home](#)

[Up](#)

[Next](#)

lynx

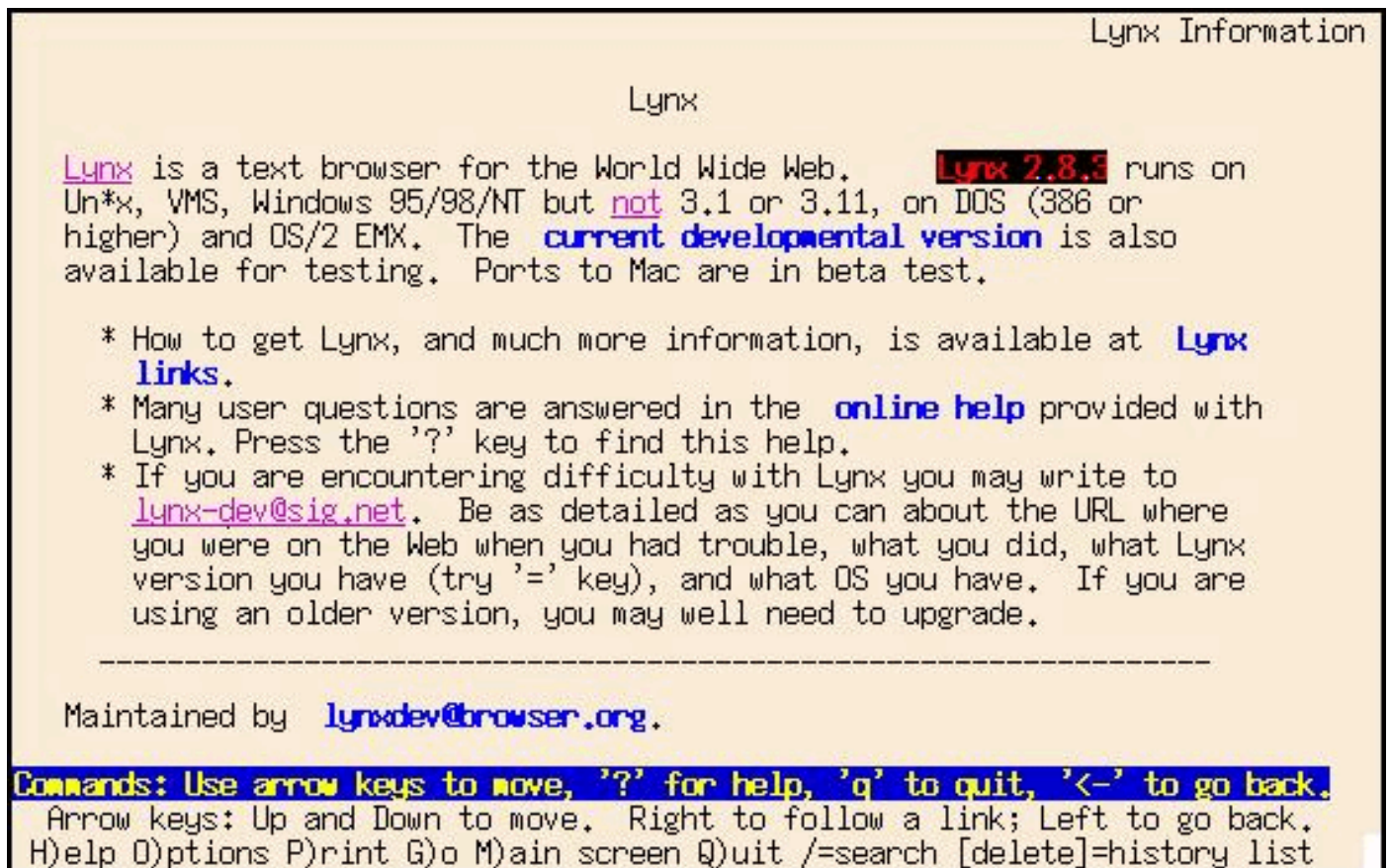
lynx

lynx(1) is a text-based web browser. It is a very quick way of looking up something on the Internet. Sometimes graphics just get in the way if you know exactly what you're after.

To start **lynx**, just type **lynx** at the prompt:

```
$ lynx
```

Figure 13-4. Lynx's default start page.



You may want to specify a site for **lynx** to open to:

```
$ lynx http://www.slackware.com
```

lynx prints the command keys and what they do at the bottom of the screen. The up and down arrow keys move around the document, Enter selects the highlighted link, and the left arrow goes back to the previous page. Typing **d** will download the currently selected file. The **g** command brings up the Go prompt, where you can give **lynx** a URL to open.

There are many other commands in **lynx**. You can either consult the man page, or type **h** to get the help screen for more information.

wget

wget(1) is a command line utility that will download files from a specified URL. It's useful for retrieving entire web sites for offline viewing, or for a more safer download of files from HTTP or FTP servers instead of Netscape. The basic syntax is:

```
$ wget <url>
```

You can also pass options. For example, this will download the Slackware web site:

```
$ wget --recursive http://www.slackware.com
```

wget will create a `www.slackware.com` directory and store the files in there, just as the site does.

wget can also download files from FTP sites; just specify an FTP URL instead of an HTTP one.

wget has many more options, which make it nice for site specific scripts (web site mirroring and so forth). The man page should be consulted for more information.

traceroute

Slackware includes the 4.4BSD **traceroute**(8) command. It's a useful network diagnostic tool. **traceroute** displays each host that a packet travels through as it tries to reach its destination. You can see how many hops from the Slackware web site you are with this command:

```
$ traceroute www.slackware.com
```

Each host will be displayed, along with the response times at each host. Here is an example output:

```
$ traceroute www.slackware.com
traceroute to www.slackware.com (204.216.27.13), 30 hops max, 40 byte packets
 1  zuul.tdn (192.168.1.1)  0.409 ms  1.032 ms  0.303 ms
 2  207.171.227.254 (207.171.227.254)  18.218 ms  32.873 ms  32.433 ms
 3  border-sf-2-0-4.sirius.com (205.134.230.254) 15.662 ms 15.731 ms 16.142 ms
 4  pb-nap.crl.net (198.32.128.20)  20.741 ms  23.672 ms  21.378 ms
 5  E0-CRL-SFO-03-E0X0.US.CRL.NET (165.113.55.3) 22.293 ms 21.532 ms 21.29 ms
 6  T1-CDROM-00-EX.US.CRL.NET (165.113.118.2)  24.544 ms  42.955 ms 58.443 ms
 7  www.slackware.com (204.216.27.13)  38.115 ms  53.033 ms  48.328 ms
```

traceroute is similar to **ping** in that it uses ICMP packets. There are several options that you can specify with **traceroute**. The default maximum number of hops is 30, but that can be changed with the **-m** option. Other options are explained in detail in the web page.

Talking to Other People

talk

talk(1) allows two users to chat. It splits the screen in half, horizontally. To request a chat with another user, use this command:

```
$ talk <person> [ttyname]
```

Figure 13-5. Two users in a talk session.



If you specify just a username, the chat request is assumed to be local, so only local users are queried. The `ttyname` is required if you want to ring a user on a specific user (if the user is logged in more than once). The required information for **talk** can be obtained from the **w**(1) command.

talk can also ring users on remote hosts. For the username you simply specify an email address. **talk** will try to contact that remote user on that host.

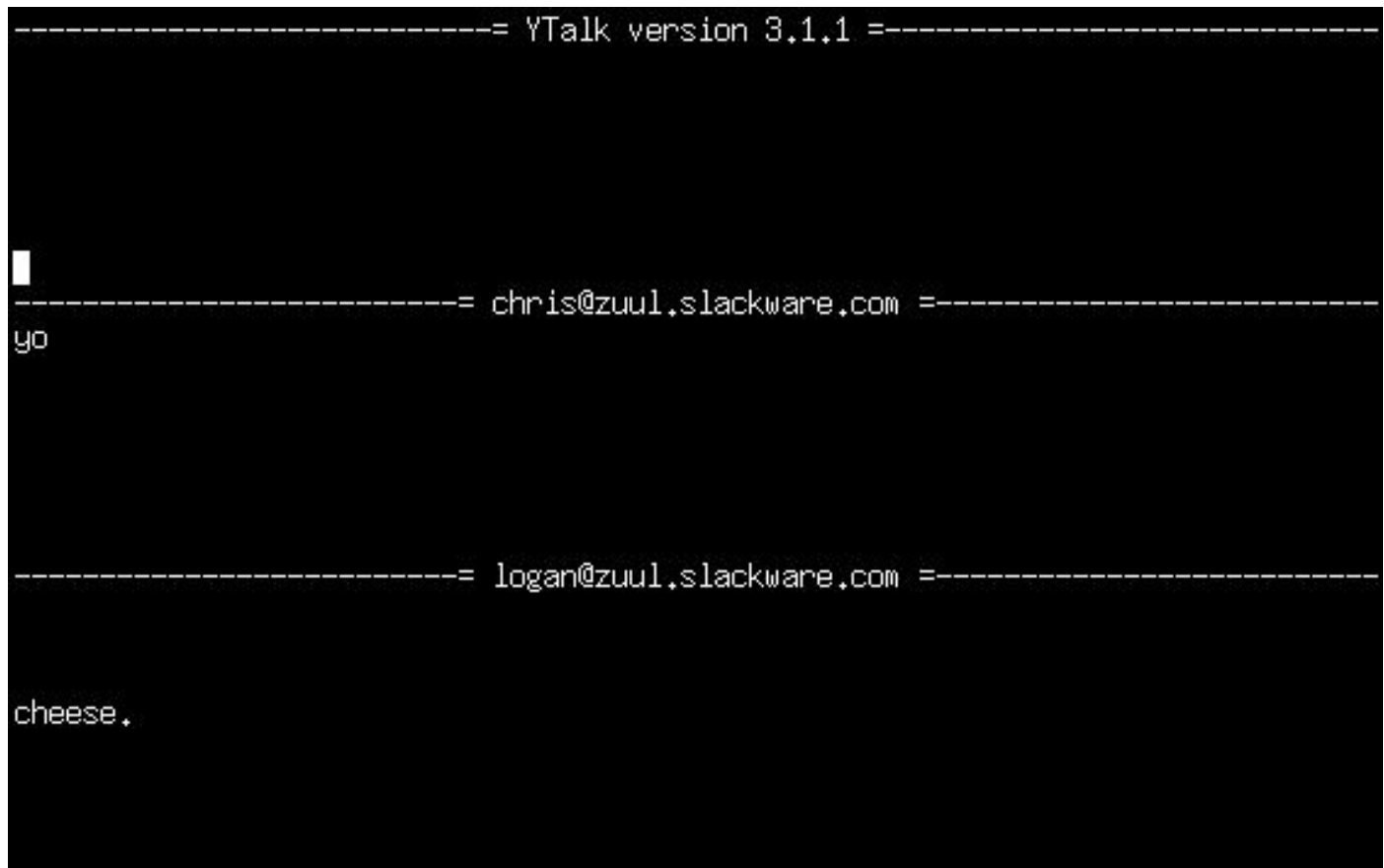
talk is somewhat limited. It only supports two users and is half-duplex.

ytalk

ytalk(1) is a backwards compatible replacement for **talk**. It comes with Slackware as the **ytalk** command. The syntax is similar, but has a few differences:

```
$ ytalk <username>[#ttyname]
```

Figure 13-6. Three users in a ytalk session.



The username and terminal are specified the same as under **talk**, except you must put them together with the hash mark (#).

ytalk offers several advantages:

- It supports more than two users.
- A menu of options that can be brought up anytime with **Esc**.
- You can shell out while still in the talk session.
- Plus more...

If you're a server administrator, you'll want to make sure that the **ntalk** port is enabled in `/etc/inetd.conf`. **ytalk** needs that to work properly.

Summary

You should now know some basic network diagnosis commands. Using these, you can determine if there is a problem with a computer or the network between your system and a remote one. Also, you should know about several mail readers, web browsers, ftp clients, and communication programs.

Chapter 14. Archive Files

Table of Contents

[gzip](#)[bzip2](#)[tar](#)[zip](#)[Summary](#)

Under Slackware Linux, there are several programs that can be used to compress and archive files. These programs are especially useful for making backups and sending copies of files between machines over a network connection. There are programs for dealing with Unix formatted archives, as well as Windows archives.

gzip

gzip(1) is the GNU compression program. It takes a single file and compresses it. The basic usage is as follows:

```
$ gzip infile
```

The resulting file will be named `infile.gz` and will usually be smaller than the input file. Note that `infile.gz` will replace `infile`. This means that `infile` will no longer exist, even though a gzipped copy will. Regular text files will compress nicely, while jpeg images, mp3s, and other such files will not compress too well as they are already compressed. This basic usage is a balance of final file size and compression time. The maximum compression can be achieved like so:

```
$ gzip -9 infile
```

This will take a longer time to compress the file, but the result will be as small as **gzip** can make it. Using lower values for the command line option will cause it to compress faster, but the file will not be as compressed.

Decompressing gzipped files can be done using two commands, which are really just the same program. **gzip** will decompress any file with a recognized file extension. A recognized extension can be any of the following: `.gz`, `-gz`, `.z`, `-z`, `.Z`, or `-Z`. The first method is to call **gunzip**(1) on a file, like so:

```
$ gunzip infile.gz
```

This will leave a decompressed version of `infile` in the current directory, and the `.gz` extension will be stripped from the filename.

The other way to decompress a gzipped file is to call **gzip** on the file:

```
$ gzip -d infile.gz
```

This will cause exactly the same behavior as calling **gunzip**. The reason for this is simple: **gunzip** is simply a symbolic link to `/bin/gzip`:

```
$ cd /usr/bin
$ ls -l gunzip
lrwxrwxrwx  1 root  root  9 Feb  2 09:45 gunzip -> /bin/gzip
```

So, running **gunzip** is really just running **gzip** with a different name. The program can determine how it is being called and take action appropriately. In this case, **gzip** will know that it is being called as **gunzip** and decompress the file. Therefore, you can use **gzip** or **gunzip** to decompress any gzipped file.

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

bzip2

bzip2

bzip2(1) is an alternative compression program installed on Slackware Linux. It uses a different compression algorithm from **gzip**, which results in some advantages and some disadvantages. The main advantage for **bzip2** is the compressed file size. **bzip2** will almost always compress better than **gzip**. In some instances, this can result in dramatically smaller files. This can be a great advantage for people on slower modem connections.

The disadvantage to **bzip2** is that it is more CPU intensive than **gzip**. This means that bzipping a file will generally take longer and will use more of the CPU than gzipping the file would. When considering which compression program to use, you must weigh this speed vs. compressed size and determine which is more important.

The usage of **bzip2** is very similar to **gzip**, so not much time will be spent discussing that. Simply call **bzip2** with a filename to compress it:

```
$ bzip2 infile
```

The resulting output file will usually be smaller than the input file, and will be called `infile.bz2`. As with **gzip**, the input file will no longer exist, since **bzip2** replaces the input file with a compressed copy.

You can also use a numeric command line argument to tweak compression rates and speed as with **gzip**. The following example shows how to achieve maximum compression with **bzip2** with considerable CPU usage:

```
$ bzip2 -9 infile
```

There are two commands to decompress files ending in a `.bz2` extension, just as with **gzip**. You can use **bzip2** or **bunzip2**(1) to decompress bzipped files. Using **bzip2** requires using a command line argument:

```
$ bzip2 -d infile.bz2
```

This will decompress the bzipped file and replace it with the decompressed copy. This resulting file will also have been stripped of the `.bz2` extension. Similarly, you can use **bunzip2** to decompress the file:

```
$ bunzip2 infile.bz2
```

You'll get the same behavior either way, thanks again to a symbolic link. Checking out `/bin/bunzip2` shows that it is simply a symbolic link to `/bin/bzip2`. This uses the same trick that **gzip** did. You'll find that calling a program using several different names to achieve different behaviors is a favorite trick of Linux programmers.

```
$ cd /bin
$ ls -l bunzip2
lrwxrwxrwx 1 root root          5 Feb  2 09:45 /bunzip2 -> bzip2
```

[Prev](#)

Archive Files

[Home](#)

[Up](#)

[Next](#)

tar

tar

tar(1) is the GNU tape archiver. It takes several files or directories and creates one large file. This allows you to compress an entire directory tree, which is impossible by just using **gzip** or **bzip2**. **tar** has many command line options, which are explained in its man page. This section will just cover the most common uses of **tar**.

The most common use for **tar** is to decompress and unarchive a package that you've downloaded from a web site or ftp site. Most files will come with a `.tar.gz` extension. This is commonly known as a `tarball`. It means that several files were archived using **tar** and then compressed using **gzip**. You might also see this listed as a `.tar.Z` file. It means the same thing, but this is usually encountered on older Unix systems.

Alternatively, you might find a `.tar.bz2` file somewhere. Kernel source is distributed as such because it is a smaller download. As you might have guessed, this is several files archived with **tar** and then bziped.

You can get to all the files in this archive by making use of **tar** and some command line arguments. Unarchiving a tarball makes use of the `-z` flag, which means to first run the file through **gunzip** and decompress it. The most common way to decompress a tarball is like so:

```
$ tar -xvzf hejaz.tar.gz
```

That's quite a few options. So what do they all mean? The `-x` means to extract. This is important, as it tells **tar** exactly what to do with the input file. In this case, we'll be splitting it back up into all the files that it came from. `-v` means to be verbose. This will list all the files that are being unarchived. It is perfectly acceptable to leave this option off, if somewhat boring. Alternatively, you could use `-vv` to be very verbose and list even more information about each file being unarchived. The `-z` option tells **tar** to run `hejaz.tar.gz` through **gunzip** first. And finally, the `-f` option tells **tar** that the next string on the command line is the file to operate on.

There are a few other ways to write this same command. On older systems lacking a decent copy of GNU tar, you might see it written like so:

```
$ gzip -dc hejaz.tar.gz | tar -xvf -
```

This command line will unzip the file and send the output to **tar**. Since **gzip** will write its output to standard out if told to do so, this command will write the decompressed file to standard out. The pipe then sends it to **tar** for unarchiving. The `-` means to operate on standard input. It will unarchive the stream of data that it gets from **gzip** and write that to the disk.

Another way to write the first command line is to leave off the dash before the options, like

so:

```
$ tar xvzf hejaz.tar.gz
```

You might also encounter a bziped archive. The version of **tar** that comes with Slackware Linux can handle these the same as gzipped archives. Instead of the **-x** command line option, you'd use **-y**:

```
$ tar -xvyf foo.tar.bz2
```

It is important to note that **tar** will place the unarchived files in the current directory. So, if you had an archive in `/tmp` that you wanted to decompress into your home directory, there are two options. First, the archive could be moved into your home directory and then run through **tar**. Or, you could specify the path to the archive file on the command line:

```
$ tar -xvzf /tmp/bar.tar.gz
```

The contents of the archive would be dumped into your home directory, and the original compressed archive file will still be in `/tmp`.

The second most common operation with **tar** is making your own archives. Making an archive is no more complicated than unarchiving other files; it just takes a different set of command lines options.

To create a compressed tar archive of all the files in the current directory (including any subdirectories and their files), you would use **tar** like so:

```
$ tar -cvzf archive.tar.gz .
```

In this command line, the **-c** option tells **tar** to create an archive, while the **-z** option runs the resulting archive file through **gzip** to compress it. `archive.tar.gz` is the file that you want to create. You can call it anything you want, and if you include a full path name, it will put the archive in that directory. Here is an example of that:

```
$ tar -cvzf /tmp/archive.tar.gz .
```

The archive would then go into `/tmp`. You can also list all the file and directories that you want to be included in the archive by listing them at the end of the command. In this case, the `.` is the directory to include in the archive. This could easily be replaced with a list of various files, or whatever you want to archive.

[Prev](#)

bzip2

[Home](#)

[Up](#)

[Next](#)

zip

zip

Finally, there are two utilities that can be used on zip files. These are very common in the Windows world, so Linux has programs to deal with them. The compression program is called **zip**(1), and the decompression program is called **unzip**(1).

Compressing one file is easy:

```
$ zip foo *
```

This will create the file `foo.zip`, which will contain all the files in the current directory. **zip** will add the `.zip` extension automatically, so there's no need to include that in the file name. You can also recurse through the current directory, zipping up any directories that are also laying around:

```
$ zip -r foo *
```

Decompressing files is easy, as well.

```
$ unzip foo
```

This will extract all the files in the file `foo.zip`, including any directories in the archive.

The **zip** utilities have several advanced options for creating self-extracting archives, leaving out files, controlling compressed file size, printing out what will happen, and much more. See the man pages for **zip** and **unzip** to find out how to use these options.

Summary

This chapter discussed the programs that are used for compressing and decompressing archive files. You should know what an archive file is, how to create one using **tar** and your choice of compression program, how to decompress one, and how to handle Windows-based archives. Almost anything that you download or upload will involve an archive, so these are important skills to know.

Chapter 15. vi

Table of Contents

[Starting vi](#)[Modes](#)[Opening Files](#)[Saving Files](#)[Quitting vi](#)[vi Configuration](#)[vi Keys](#)[Summary](#)

vi(1) is the standard Unix text editing program, and mastering it is essential for system administrators. There are several versions (or clones) of **vi** available, including **vi**, **elvis**, **vile**, and **vim**. One of these is available on just about any version of Unix, as well as on Linux. All of these versions include the same basic feature set and commands, so learning one clone should make it easy to learn another.

vi includes a number of powerful features including syntax highlighting, code formatting, a powerful search-and-replace mechanism, macros, and more. These features make it especially attractive to programmers, web developers, and the like. System administrators will appreciate the automation and integration with the shell that is possible.

On Slackware Linux, the default version of **vi** available is **elvis**. Other versions - including **vim** and **gvim** - are available if you've installed the proper packages. **gvim** is an X Window version of **vim** that includes toolbars, detachable menus, and dialog boxes.

Starting vi

vi can be started from the command line in a variety of ways. The simplest form is just:

```
$ vi
```

Figure 15-1. A vi session.

```

-----
Mon Jun  5 00:58:56 PDT 2000
Added KDE 1.90 (code named Konfucious), a beta preview of KDE's next
generation desktop, including the new KOffice suite. It all looks really
nice, and I can't wait to see the finished 2.0 release. :)
contrib/kde-1.90/kde-i18n.tgz: KDE 1.90 Internationalization support.
contrib/kde-1.90/kde-qt-addon.tgz: Qt add-on needed by KDE.
contrib/kde-1.90/kdebase.tgz: KDE 1.90 base package.
contrib/kde-1.90/kdegames.tgz: KDE 1.90 games.
contrib/kde-1.90/kdelibs.tgz: KDE 1.90 system libraries.
contrib/kde-1.90/kdenetwork.tgz: KDE 1.90 network apps.
# See "man 8 inetd" for more information.
#
# If you make changes to this file, either reboot your machine or send the
# inetd a HUP signal:
# Do a "ps x" as root and look up the pid of inetd. Then do a
# "kill -HUP <pid of inetd>".
# The inetd will re-read this file whenever it gets that signal.
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
# The first 4 services are really only used for debugging purposes, so
# we comment them out since they can otherwise be used for some nasty
13 rows, 80 columns

```

This will start up **vi** with an empty buffer. At this point, you'll see a mostly blank screen. It is now in command mode, waiting for you to do something. For a discussion of the various **vi** modes, see [the section called Modes](#). In order to quit out of **vi**, type the following:

```
:q
```

Assuming that there have been no changes to the file, this will cause **vi** to quit. If there have been changes made, it will warn you that there have been changes and tell you how to disregard them.

You can also start **vi** with a pre-existing file. For example, the file `/etc/resolv.conf` would be opened like so:

```
$ vi /etc/resolv.conf
```

Finally, **vi** can be started on a particular line of a file. For example, you could start up **vi** on line 47 of `/usr/src/linux/init/main.c` like so:

```
vi +47 /usr/src/linux/init/main.c
```

vi will display the given file and will place the cursor at the specified line. In the case where you specify a line that is after the end of the file, **vi** will place the cursor on the last line. This is especially helpful for programmers, as they can jump straight to the location in the file that an error occurred, without having to search for it.

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

Modes

Modes

vi operates in various modes, which are used to accomplish various tasks. When you first start **vi**, you are placed into command mode. From this point, you can issue various commands to manipulate text, move around in the file, save, quit, and more. Editing the text is done in insert mode. You can quickly move between modes with a variety of keystrokes, which are explained below.

Command Mode

You are first placed into command mode. From this mode, you cannot directly enter text or edit what is already there. However, you can manipulate the text, search, quit, save, load new files, and more. This is intended only to be an introduction to the command mode. For a description of the various commands, see [the section called vi Keys](#).

Probably the most often used command in command mode is changing to insert mode. This is accomplished by hitting the **i** key. The cursor changes shapes, and `-- INSERT --` is displayed at the bottom of the screen (note that this does not happen in all clones of **vi**). From there, all your keystrokes are entered into the current buffer and are displayed to the screen. To get back into command mode, hit the escape key.

Command mode is also where you move around in the file. On some systems, you can use the arrow keys to move around. On other systems, you may need to use the more traditional keys of `hjkl`. Here is a simple listing of how these keys are used to move around:

h	move left one character
j	move down one character
k	move up one character
l	move right one character

Simply press a key to move. As you will see later, these keys can be combined with a number to move much more efficiently.

Many of the commands that you will use in command mode begin with a colon. For example, quitting is **:q**, as discussed earlier. The colon simply indicates that it is a command, while the **q** tells **vi** to quit. Other commands are an optional number, followed by a letter. These commands do not have a colon before them, and are generally used to manipulate the text.

For example, deleting one line from a file is accomplished by hitting **dd**. This will remove the line that the cursor is on. Issuing the command **4dd** would tell **vi** to remove the line that the cursor is on and the three after that. In general, the number tells **vi** how many times to

perform the command.

You can combine a number with the movement keys to move around several characters at a time. For example, **10k** would move up ten lines on the screen.

Command mode can also be used to cut and paste, insert text, and read other files into the current buffer. Copying text is accomplished with the **y** key (y stands for yank). Copying the current line is done by typing **yy**, and this can be prefixed with a number to yank more lines. Then, move to the location for the copy and hit **p**. The text is pasted on the line after the current one.

Cutting text is done by typing **dd**, and **p** can be used to paste the cut text back into the file. Reading in text from another file is a simple procedure. Just type **:r**, followed by a space and the file name that contains the text to be inserted. The file's contents will be pasted into the current buffer on the line after the cursor. More sophisticated **vi** clones even contain filename completion similar to the shell's.

The final use that will be covered is searching. Command mode allows for simple searching, as well as complicated search-and-replace commands that make use of a powerful version of regular expressions. A complete discussion of regular expressions is beyond the scope of this chapter, so this section will only cover simple means of searching.

A simple search is accomplished by hitting the **/** key, followed by the text that you are searching for. **vi** will search forward from the cursor to the end of the file for a match, stopping when it finds one. Note that inexact matches will cause **vi** to stop as well. For example, a search for `the` will cause **vi** to stop on `then`, `therefore`, and so on. This is because all of those words do match `the`, but only at the beginning.

After **vi** has found the first match, you can continue on to the next match simply by hitting the **/** key followed by enter. You can also search backwards through the file by replacing the slash with the **?** key. For example, searching backwards through the file for `the` would be accomplished by typing **?the**.

Insert Mode

Inserting and replacing text is accomplished in insert mode. As previously discussed, you can get into insert mode by hitting **i** from command mode. Then, all text that you type is entered into the current buffer. Hitting the escape key takes you back into command mode.

Replacing text is accomplished in several ways. From command mode, hitting **r** will allow you to replace the one character underneath the cursor. Just type the new character and it will replace the one under the cursor. You will then be immediately placed back into command mode. Hitting **R** allows you to replace as many characters as you'd like. To get out of this replacement mode, just hit escape to go back into command mode.

There is yet another way to toggle between insertion and replacement. Hitting the insert key from command mode will take you into insert mode. Once you are in insert mode, the keyboard's Insert key serves as a toggle between insert and replace. Hitting it once will allow you to replace. Hitting it once more will once again allow you to insert text.

[Prev](#)

vi

[Home](#)

[Up](#)

[Next](#)

Opening Files

Opening Files

vi allows you to open files from command mode as well as specifying a file on the command line to open. To open the file `/etc/lilo.conf`:

```
:e /etc/lilo.conf
```

If you have made changes to the current buffer without saving, **vi** will complain. You can still open the file without saving the current buffer by typing **:e!**, followed by a space and the filename. In general, **vi**'s warnings can be suppressed by following the command with an exclamation mark.

If you want to reopen the current file, you can do so simply by typing **e!**. This is useful if you have somehow messed up the file and want to reopen it.

Some **vi** clones (for example, **vim**) allow for multiple buffers to be open at the same time. For example, to open up the file `09-vi.sgml` in my home directory while another file was open, I would type:

```
:split ~/09-vi.sgml
```

The new file is displayed on the top half of the screen, and the old file is displayed in the bottom half of the screen. There are a lot of commands that manipulate the split screen, and many of these commands start to resemble something out of **EMACS**. The best place to look up these commands would be the man page for your **vi** clone. Note that many clones do not support the split-screen idea, so you might not be able to use it at all.

Saving Files

There are several ways to save files in **vi**. If you want to save the current buffer to the file `randomness`, you would type:

```
:w randomness
```

Once you've saved the file once, saving it again is as simple as typing **:w**. Any changes will be written out to the file. After you've saved the file, you are dumped back into command mode. If you want to save the file and quit **vi** (a very common operation), you would type **:wq**. That tells **vi** to save the current file and quit back to the shell.

On occasion, you want to save a file that is marked as read-only. You can do this by adding an exclamation point after the write command, like so:

```
:w!  
:wq!
```

However, there will still be instances where you cannot write the file (for example, you are attempting to edit a file that is owned by another user). When this happens, **vi** will tell you that it cannot save the file. If you really want to edit the file, you'll have to come back and edit it as root.

Quitting vi

One way to quit **vi** is through **:wq**, which will save the current buffer before quitting. You can also quit without saving with **:q** or **:q!**. The latter is used when you've modified the file but have not made any changes to it.

On occasion, your machine might crash or **vi** might crash. However, both **elvis** and **vim** will take steps to minimize the damage to any open buffers. Both editors save the open buffers to a temporary file on occasion. This file is usually named similarly to the open file, but with a dot at the beginning. This makes the file hidden.

This temporary file gets removed once the editor quits under normal conditions. This means that the temporary copy will still be around if something crashes. When you go back to edit the file again, you will be prompted for what action to take. In most cases, a large amount of your unsaved work can be recovered. **elvis** will also send you a mail (from Graceland, oddly enough :) telling you that a backup copy exists.

vi Configuration

Your **vi** clone of choice can be configured in several ways.

A variety of commands can be entered while in command mode to set up **vi** just how you like it. Depending on your editor, you can enable features to make programming easier (like syntax highlighting, auto-indenting, and more), set up macros to automake tasks, enable textual substitutions, and more.

Almost all of these commands can be put into a configuration file in your home directory. **elvis** expects a `.exrc` file, while **vim** expects a `.vimrc` file. Most of the setup commands that can be entered in command mode can be placed in the configuration file. This includes setup information, textual substitutions, macros, and more.

Discussing all these options and the differences between the editors is quite an involved subject. For more information, check out the man page or web site for your preferred **vi** editor. Some editors (like **vim**) have extensive help within the editor that can be accessed with the `:help` command, or something similar. You can also check out the O'Reilly book *Learning the vi Editor* by Lamb and Robbins.

Many common programs in Linux will load up a text file in **vi** by default. For example, editing your crontabs (see the section on cron) will start up **vi** by default. If you do not like **vi** and would like another editor to be started instead, all you need to do is set the `VISUAL` environment variable to the editor you prefer. For information on setting environment variables, see [the section called *Environment Variables* in Chapter 8](#). If you want to make sure that your editor will be the default every time you login, add the `VISUAL` setting to your `.bash_profile` or `.bashrc` files.

vi Keys

This section is a quick reference of many common vi commands. Some of these were discussed earlier in the chapter, while many will be new.

Table 15-1. Movement

Operation	Key
left, down, up, right	h, j, k, l
To the end of the line	\$
To the beginning of the line	^
To the end of the file	G
To the beginning of the file	:1
To line 47	:47

Table 15-2. Editing

Operation	Key
Removing a line	dd
Removing five lines	5dd
Replacing a character	r
Removing a character	x
Removing ten characters	10x
Undo last action	u
Join current and next lines	J

Table 15-3. Searching

Operation	Key
Search for asdf	/asdf
Search backwards for asdf	?asdf
Repeat last search forwards	/
Repeat last search backwards	?

Table 15-4. Saving and Quitting

Operation	Key
Quit	:q
Quit without saving	:q!
Write and quit	:wq
Write, without quitting	:w

Reload currently open file **:e!**
Write buffer to file asdf **:w asdf**
Open file hejaz **:e hejaz**
Read file asdf into buffer **:r asdf**
Read output of **ls** into buffer **:r !ls**

[Prev](#)

vi Configuration

[Home](#)

[Up](#)

[Next](#)

Summary

Summary

You should now have some basic familiarity with **vi** - the standard Unix text editor. **vi** is a fairly complex program with lots of commands and configuration options. However, you should be able to open a file, move around, edit the file, and quit. This is all you will have to do for most day-to-day operations. As you find the need for more power, you can use **vi**'s extensive help to learn about it.

Chapter 16. Slackware Package Management

Table of Contents

[Overview of Package Format](#)

[Package Utilities](#)

[Making Packages](#)

[Making Tags and Tagfiles \(for setup\)](#)

[Summary](#)


A software package is a bundle of related programs that are ready for you to install. When you download a source code archive, you have to configure, compile, and install it by hand. With a software package, this has already been done for you. All that you have to do is install the package. Another handy feature of using software packages is that it is very easy to remove and upgrade them, if you so desire. Slackware comes with programs for all your package management needs. You can install, remove, upgrade, make, and examine packages very easily.

Overview of Package Format

Before learning the utilities, you should become familiar with the format of a Slackware package. A package is simply a tar archive file that has been compressed with **gzip**. A package is built so that it can be extracted in the root filesystem.

Here is a fictitious program and its example package:

```
./
usr/
usr/bin/
usr/bin/makehejaz
usr/doc/
usr/doc/makehejaz-1.0/
usr/doc/makehejaz-1.0/COPYING
usr/doc/makehejaz-1.0/README
usr/man/
usr/man/man1
usr/man/man1/makehejaz.1.gz
install/
install/doinst.sh
```



The package system will extract this file in the root directory to install it. An entry in the package database will be created that contains the contents of this package so that it can be upgraded or removed later.

Notice the `install/` subdirectory. This is a special directory that can contain a postinstallation script called `doinst.sh`. If the package system finds this file, it will execute it after installing the package.

Other scripts can be embedded in the package, but those are discussed more in detail in [the section called *makepkg*](#).

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

Package Utilities

Package Utilities

There are four main utilities for package management. They perform installation, removal, and upgrades of packages.

pkgtool

pkgtool(8) is a menu-driven program that allows installation and removal of packages. The main menu looks like this:

Figure 16-1. Pkgtool's main menu.



Installation is offered from the current directory, another directory, or from floppy disks. Simply select the installation method you want and **pkgtool** will search that location for valid packages to install.

You may also view a list of installed packages, which would look like this:

Figure 16-2. Pkgtool view mode.



If you want to remove packages, select the remove option and you will be presented with a checklist of all the installed packages. Flag the ones you want to remove and select OK. **pkgtool** will remove them.

Some users prefer this utility to the command line utilities. However, it should be noted that the command line utilities offer many more options. Also, the ability to upgrade packages is only offered through the command line utilities.

installpkg

installpkg(8) handles installation of new packages on the system. The syntax is as follows:

```
# [ROOT=<path>] installpkg [option] <package name>...
```

Three options are provided for **installpkg**. Only one option can be used at a time.

Table 16-1. installpkg Options

Option	Effects
-m	Performs a makepkg operation on the current directory.
-warn	Shows what would happen if you installed the specified package. This is useful for production systems so you can see exactly what would happen before installing something.
-r	Recursively install all packages in the current directory and down. The <package name> can use wildcards, which would be used as the search mask when recursively installing.

If you pass the **ROOT** environment variable before **installpkg**, that path will be used for the root directory. This is useful for setting up new drives for your root directory. They will typically be mounted to `/mnt` or something other than `/`.

The installed package database entry is stored in `/var/log/packages`. The entry is really just a plain text file, one for each package. If the package has a postinstallation script, it is written to `/var/log/scripts/<packagename>`.

You may specify several packages or use wildcards for the package name. Be advised that **installpkg** will not tell you if you are overwriting an installed package. It will simply install right on top of the old one. If you want to ensure that old files from the previous package are safely removed, use **upgradepkg**.

removepkg

removepkg(8) handles removing installed packages from the system. The syntax is as follows:

```
# [ROOT=<path>] removepkg [option] <package name>...
```

Four options are provided for **removepkg**. Only one option may be used at a time.

Table 16-2. removepkg Options

Option	Effects
-copy	The package is copied to the preserved packages directory. This creates a tree of the original package without removing it.
-keep	Saves temporary files created during the removal. Really only useful for debugging purposes.
-preserve	The package is removed, but copied to the preserved packages directory at the same time.
-warn	Shows what would happen if you removed the package.

If you pass the **ROOT** environment variable before **removepkg**, that path will be used for the root directory. This is useful for setting up new drives for your root directory. They will typically be mounted to `/mnt` or something other than `/`.

removepkg looks at the other installed packages and only removes files unique to the package you specify. It will also scan the postinstallation script for the specified package and remove any symbolic links that were created by it.

During the removal process, a status report is displayed. After the removal, the package database entry is moved to `/var/log/removed_packages` and the postinstallation script is moved to `/var/log/removed_scripts`.

Just as with **installpkg**, you can specify several packages or use wildcards for the package name.

upgradepkg

upgradepkg(8) will upgrade an installed Slackware package. The syntax is as follows:

```
# [ROOT=<path>] upgradepkg <package name>...
```

or

```
# [ROOT=<path>] upgradepkg \  
<old package name>%<new package name>
```

upgradepkg works by first installing the new package and then removing the old package so that old files are no longer around on the system. If the upgraded package name has changed, use the percent sign syntax to specify the old package (the one that is installed) and the new package (the one you are upgrading it to).

If you pass the **ROOT** environment variable before **upgradepkg**, that path will be used for the root directory. This is useful for setting up new drives for your root directory. They will typically be mounted to `/mnt` or something other than `/`.

upgradepkg is not flawless. You should always back up your configuration files. If they get overwritten, you'll still have a copy of the originals for any needed repair work.

Just as with **installpkg** and **removepkg**, you can specify several packages or use wildcards for the package name.

rpm2tgz/rpm2targz

The Red Hat Package Manager is a popular packaging system available today. Many software distributors are offering their products in RPM format. Since this is not our native format, we do not recommend people rely on them. However, some things are only available as an RPM (even the source).

We provide a program that will convert RPM packages to our native .tgz format. This will allow you to extract the package (perhaps with **explodepkg**) to a temporary directory and examine its contents.

The **rpm2tgz** program will create a Slackware package with a .tgz extension, while **rpm2targz** creates an archive with a .tar.gz extension.

[Prev](#)

Slackware Package
Management

[Home](#)

[Up](#)

[Next](#)

Making Packages

Making Packages

Making Slackware packages can be both easy and difficult. There is no specific method for building a package. The only requirement is that the package be a tar gzipped file and if there is a postinstallation script, it must be `/install/doinst.sh`.

If you are interested in making packages for your system or for a network that you manage, you should have a look at the various build scripts in the Slackware source tree. There are several methods we use for making packages.

explodepkg

explodepkg(8) will do the same thing that **installpkg** does to extract the package, but it doesn't actually install it and it doesn't record it in the packages database. It simply extracts it to the current directory.

If you look at the Slackware source tree, you will see how we use this command for framework packages. These packages contain a skeleton of what the final package will look like. They hold all the necessary filenames (zero-length), permissions, and ownerships. The build script will **cat** the package contents from the source directory to the package build directory.

makepkg

makepkg(8) will package up the current directory into a valid Slackware package. It will search the tree for an symbolic links and add a creation block to the postinstallation script for creating them during the package install. It also warns of any zero-length files in the package tree.

This command is typically run after you have created your package tree.

Making Tags and Tagfiles (for setup)

The Slackware setup program handles installation of the software packages on your system. There are files that tell the setup program which packages must be installed, which ones are optional, and which ones are selected by default by the setup program.

A tagfile is in the first software series directory and is called `tagfile`. It lists the packages in that particular disk set and their status. The status can be:

Table 16-3. tagfile Status Options

Option	Meaning
ADD	The package is required for proper system operation
SKP	The package will be automatically skipped
REC	The package is not required, but recommended
OPT	The package is optional

The format is simply:

```
<package name>: <status>
```

One package per line. The original tagfiles for each software series are stored as `tagfile.org`. So if you mess up yours, you can restore the original one.

Many administrators prefer writing their own tagfiles and starting the installer and selecting full . The setup program will read the tagfiles and perform the installation according to their contents. If you use REC or OPT, a dialog box will be presented to the user asking whether or not they want a particular package. Therefore, it is recommended that you stick with ADD and SKP when writing tagfiles for automated installs.

Just make sure your tagfiles are written to the same location as the originals. Or you can specify a custom tagfile path if you have custom tagfiles.

Summary

You should now be familiar with the idea of a software package and how these are used in Slackware. You should be familiar with the various package management utilities and how to use them. The most important parts of this chapter are how to install, remove, and upgrade packages. That's the most common use of the package utilities. However, you should also have some understanding of making and inspecting packages.

Chapter 17. ZipSlack and BigSlack

Table of Contents

[What is ZipSlack/BigSlack?](#)

[Getting ZipSlack/BigSlack](#)

[Installation](#)

[Booting ZipSlack/BigSlack](#)

[Adding, Removing, and Upgrading Software](#)

[Common Problems](#)

[Getting Help](#)

[Summary](#)

What is ZipSlack/BigSlack?

ZipSlack is a special version of Slackware Linux. It's an already installed copy of Slackware that's ready to run from your DOS or Windows partition. It's a basic installation, you do not get everything that comes with Slackware. If you want everything with ZipSlack, then you should try BigSlack .

ZipSlack gets its name from the form it's distributed in, a big .ZIP file. Users of DOS and Windows will probably be familiar with these files. They are compressed archives. The ZipSlack archive contains everything you need to get up and running with Slackware.

It is important to note that ZipSlack and BigSlack are significantly different than a regular installation. Even though they function the same and contain the same programs, their intended audiences and functions differ. Several advantages and disadvantages of ZipSlack and BigSlack are discussed below.

One last thing, you should always review the documentation included in the actual ZipSlack or BigSlack directory. It contains the latest information regarding installation, booting, and general use of the product.

Advantages

- Does not require repartitioning of your hard disk.
- Great way to learn Slackware Linux without stumbling through the installation process.

Disadvantages

- Uses the DOS filesystem, which is slower than a native Linux filesystem.
- Will not work with Windows NT.

[Prev](#)

Summary

[Home](#)

[Up](#)

[Next](#)

Getting ZipSlack/BigSlack

Getting ZipSlack/BigSlack

Obtaining ZipSlack or BigSlack is easy. If you have purchased the official Slackware Linux CD set, then you already have ZipSlack and BigSlack. Just find the CD that contains the one you want and place it in your CD-ROM drive. It's usually the third or fourth disc, but always trust the labels over this documentation.

If you want to download ZipSlack or BigSlack, you should first visit our [Get Slack](#) web page for the latest download information:

<http://www.slackware.com/getslack/>

ZipSlack and BigSlack are part of each Slackware release. Locate the release you want, and go to that directory on the FTP site. The latest release directory can be found at this location:

<ftp://ftp.slackware.com/pub/slackware/slackware/>

You'll find ZipSlack in the `/zipslack` subdirectory and BigSlack in the `/bigslack` directory. ZipSlack is offered as one big `.ZIP` file or floppy-sized chunks. The chunks are in the `/zipslack/split` directory. BigSlack is only offered in chunk form.

Don't stop at just the `.ZIP` files. You should also download the documentation files and any boot images that appear in the directory.

Installation

Once you've downloaded the necessary components, you'll need to extract the .ZIP file (or files if you downloaded the chunks). Be sure to use a 32-bit unzipper. The size and filenames in the archive are too much for a 16-bit unzipper. Examples of 32-bit unzippers include WinZip and PKZIP for Windows.

Both ZipSlack and BigSlack are designed to be extracted directly to the root directory of a drive (such as C: or D:). A \LINUX directory will be created that contains the actual Slackware installation. You'll also find the files necessary to booting the system in that directory as well.

After you've extracted the files, you should have a \LINUX directory on the drive of your choosing (we'll use C: from here on).

Booting ZipSlack/BigSlack

There are several ways to boot ZipSlack and BigSlack. The most common is to use the included `LINUX.BAT` to boot the system from DOS (or from DOS mode under Windows 9x). This file must be edited to match your system before it will work.

Start by opening the `C:\LINUX\LINUX.BAT` file in your favorite text editor. At the top of the file you will notice a large comment. It explains what you need to edit in this file (and also what to do if you are booting from an external Zip drive). Don't worry if you don't understand the `root=` setting. There are several examples, so feel free to pick one and try it. If it doesn't work, you can edit the file again, comment out the line you uncommented, and pick another one.

After you uncomment the line you want by removing the `rem` at the beginning of the line, save the file and exit the editor. Bring your machine into DOS mode.



A DOS prompt window in Windows 9x will NOT work.

Type **C:\LINUX\LINUX.BAT** to boot the system. If all goes well, you should be presented with a login prompt.

Log in as **root**, with no password. You'll probably want to set a password for root, as well as adding an account for yourself. At this point you can refer to the other sections in this book for general system usage.

If using the `LINUX.BAT` file to boot the system didn't work for you, you should refer to the included `C:\LINUX\README.1ST` file for other ways to boot.

Adding, Removing, and Upgrading Software

ZipSlack and BigSlack can use the same packages that a regular Slackware installation uses. That means you can use the standard Slackware package tools to add software, remove software, and upgrade software. You can even add packages right off the Slackware CD-ROM.

Refer to [Chapter 16](#) for more information.

Common Problems

These are the most common problems that users of ZipSlack and BigSlack run into. We offer several other methods of help if your problem isn't listed here. The next section discusses those in detail.

Unable to open initial console

This is typically caused by specifying the incorrect device partition for the `root=` setting in the `LINUX.BAT` file. Edit the `LINUX.BAT` file again and choose another `root=` partition. If you have absolutely no idea what this means, you can simply try each one until you get one that works.

This can also be caused by unpacking the `.ZIP` files in a location other than a root directory of a drive. The archives need to be extracted directly to a drive and not a subdirectory.

Kernel panic: VFS : Unable to mount root fs

This means you've specified the wrong device for the `root=` setting in the `LINUX.BAT` file. You need to edit that file again and choose another `root=` device. If you don't know which one it is, just try all of them until you find one that works.

Getting Help

Before asking for help, you should read through the documentation in the `C:\LINUX` directory to see if there is an answer to your question.

If you've read the documentation and still have problems, the methods below will hopefully get your problem resolved.

ZipSlack FAQ

The list of common questions and their answers can be found in both the `FAQ.TXT` file in the `C:\LINUX` directory as well as on the web site:

<http://www.slackware.com/faq/>

ZipSlack Discussion Forum

The online discussion forum for ZipSlack lets you talk to other ZipSlack and BigSlack users. You can post your questions here and also offer your help to other users. It's a great way to get direct help from other users.

<http://www.slackware.com/forum/>

Support Via Email

The Slackware Support Team will try to help you if you are having problems with ZipSlack or BigSlack. Be sure to clearly state the problem and provide any information that you think is necessary to solving the problem.

<support@slackware.com>

Summary

You should understand what ZipSlack and BigSlack are. If you've decided that you want to use either, you should know how to install, boot, troubleshoot, and get help. ZipSlack and BigSlack can be very convenient if you just want to try out Slackware, but don't want to remove your existing Windows partitions.

Glossary

Account

All of the information about a user, including username, password, finger information, UID and GID, and home directory. To create an account is to add and define a user.

Background

Any process that is running without accepting or controlling the input of a terminal is said to be running in the background.

Boot disk

A floppy disk containing an operating system (in our case, the Linux kernel) from which a computer can be started.

Compile

To convert source code to machine-readable binary code.

Daemon

A program designed to run in the background and, without user intervention, perform a specific task (usually providing a service).

Darkstar

The default hostname in Slackware; your computer will be called darkstar if you do not specify some other name.

One of Patrick Volkerding's development machines, named after Dark Star, a song by the Grateful Dead.

Desktop Environment

A graphical user interface (GUI) that runs atop the X Window System and provides such features as integrated applications, cohesive look-and-feel between programs and components, file and window management capabilities, etc. A step beyond the simple window manager.

Device driver

A chunk of code in the kernel that directly controls a piece of hardware.

Device node

A special type of file in the `/dev` filesystem that represents a hardware component to the operating system.

DNS

Domain Name Service. A system in which networked computers are given names which translate to numerical addresses.

Domain name

A computer's DNS name, excluding its host name.

Dot file

In Linux, files which are to be hidden have filenames beginning with a dot ('.').

Dotted quad

The format of IP addresses, so called because it consists of four numbers (range 0-255 decimal) separated by periods.

Dynamic loader

When programs are compiled under Linux, they usually use pieces of code (functions) from external libraries. When such programs are run, those libraries must be found and the required functions loaded into memory. This is the job of the dynamic loader.

Environment variable

A variable set in the user's shell which can be referenced by that user or programs run by that user within that shell. Environment variables are generally used to store preferences and default parameters.

Epoch

A period of history; in Unix, The Epoch begins at 00:00:00 UTC January 1, 1970. This is considered the dawn of time by Unix and Unix-like operating systems, and all other time is calculated relative to this date.

Filesystem

A representation of stored data in which files of data are kept organized in directories. The filesystem is the nearly universal form of representation for data stored to disks (both fixed and removable).

Foreground

A program that is accepting or controlling a terminal's input is said to be running in the foreground.

Framebuffer

A type of graphics device; in Linux, this most often refers to the software framebuffer, which provides a standard framebuffer interface to programs while keeping specific hardware drivers hidden from them. This layer of abstraction frees programs of the need to speak to various hardware drivers.

FTP

The File Transfer Protocol. FTP is a very popular method of transferring data between computers.

Gateway

A computer through which data on a network is transferred to another network.

GID

Group Identifier. The GID is a unique number attributed to a group of users.

Group

Users in Unix belong to groups, which can contain many other users and are used for more general access control than the existence of users alone can easily allow.

GUI

Graphical User Interface. A software interface that uses rendered graphical elements such as buttons, scrollbars, windows, etc. rather than solely text-based input and output

Home directory

A user's home directory is the directory the user is placed in immediately upon logging in. Users have full permissions and more or less free reign within their home directories.

HOWTO

A document describing how to do something, such as configure a firewall or manage users and groups. There is a large collection of these documents available from the Linux Documentation Project.

HTTP

The Hypertext Transfer Protocol. HTTP is the primary protocol on which the World Wide Web operates.

ICMP

Internet Control Message Protocol. A very basic networking protocol, used mostly for pings.

Kernel

The heart of an operating system. The kernel is the part that provides basic process control and interfaces with the computer's hardware.

Kernel module

A piece of kernel code, usually a driver of some sort, that can be loaded and unloaded from memory separately from the main body of the kernel. Modules are handy when upgrading drivers or testing kernel settings, because they can be loaded and unloaded without rebooting.

Library

A collection of functions which can be shared between programs.

LILLO

The LInux LOader. LILO is the most widely-used Linux boot manager.

LOADLIN

LOADLIN is a program that runs under MS DOS or Windows and boots a Linux system. It is most commonly used on computers with multiple operating systems (including Linux and DOS/Windows, of course).

Man section

Pages in the standard Unix online manual ("man") are grouped into sections for easy reference. All C programming pages are in section 3, system administration pages in section 5, etc.

MBR

The Master Boot Record. A reserved space on a hard drive where information on what to do when booting is stored. LILO or other boot managers can be written here.

Motif

A popular programming toolkit used in many older X programs.

MOTD

Message of the Day. The motd (stored in Linux in `/etc/motd`) is a text file that is displayed to all users upon logging in. Traditionally, it is used by the system administrator as a sort of bulletin board for communicating with users.

Mount point

An empty directory in a filesystem where another filesystem is to be mounted, or grafted on.

Nameserver

A DNS information server. Nameservers translate DNS names to numerical IP addresses.

Network interface

A virtual representation of a network device provided by the kernel. Network interfaces allow users and programs to talk to network devices.

NFS

The Network Filesystem. NFS allows the mounting of remote filesystems as if they were local to your computer and thus provides a transparent method of file sharing.

Octal

Base-8 number system, with digits 0-7.

Pager

An X program that allows the user to see and switch between multiple desktops.

Partition

A division of a hard drive. Filesystems exist on top of partitions.

PPP

Point-to-Point Protocol. PPP is used mainly for connecting via modem to an Internet Service Provider.

Process

A running program.

Root directory

Represented as `/`, the root directory exists at the top of the filesystem, with all other directories branching out beneath it in a file tree.

Root disk

The disk (usually fixed) on which the root directory is stored.

Routing table

The set of information the kernel uses in routing network data around. It contains such tidbits as where your default gateway is, which network interface is connected to which network, etc.

Runlevel

The overall system state as defined by `init`. Runlevel 6 is rebooting, runlevel 1 is single user mode, runlevel 4 is an X login, etc. There are 6 available runlevels on a Slackware system.

Secure shell

An encrypted (thus secure) method of logging in remotely to a computer. Many secure shell programs are available; both a client and server are needed.

Service

The sharing of information and/or data between programs and computers from a single server to multiple clients. HTTP, FTP, NFS, etc. are services.

Shadow password suite

The shadow password suite allows encrypted passwords to be hidden from users, while the rest of the information in the `/etc/passwd` file remains visible to all. This helps prevent brute-force attempts at cracking passwords.

Shell

Shells provide a commandline interface to the user. When you're looking at a text prompt, you're in a shell.

Shell builtin

A command built into the shell, as opposed to being provided by an external program. For instance, **bash** has a **cd** builtin.

Signal

Unix programs can communicate between each other using simple signals, which are enumerated and usually have specific meanings. **kill -l** will list the available signals.

SLIP

Serial Line Interface Protocol. SLIP is a similar protocol to PPP, in that it's used for connecting two machines via a serial interface.

Software package

A program and its associated files, archived and compressed into a single file along with any necessary scripts or information to aid in managing the installation, upgrade, and removal of those files.

Software series

A collection of related software packages in Slackware. All KDE packages are in the `kde` series, networking packages in the `n` series, etc.

Source code

The (more or less) human-readable code in which most programs are written. Source code is compiled into `binary` code.

Standard Error (stderr)

The Unix-standard output stream for errors. Programs write any error messages on `stderr`, so that they can be separated from normal output.

Standard Input (stdin)

The Unix-standard input stream. Data can be redirected or piped into a program's `stdin` from any source.

Standard Output (stdout)

The Unix-standard output stream. Normal text output from a program is written to `stdout`, which is separate from the error messages reported on `stderr` and can be piped or redirected into other programs' `stdin` or to a file.

Subnet

An IP address range that is part of a larger range. For instance, 192.168.1.0 is a subnet of 192.168.0.0 (where 0 is a mask meaning `undefined`); it is, in fact, the `.1` subnet.

Superblock

In Linux, partitions are discussed in terms of blocks. A block is 512 bytes. The superblock is the first 512 bytes of a partition.

Supplemental disk

In Slackware, a floppy disk used during installation that contains neither the kernel (which is on the boot disk) nor the root filesystem (which is on the root disk), but additional needed files such as network modules or PCMCIA support.

Suspended process

A process which has been frozen until killed or resumed.

Swap space

Disk space used by the kernel as virtual RAM. It is slower than RAM, but because disk space is cheaper, swap is usually more plentiful. Swap space is useful to the kernel for holding lesser-used data and as a fallback when physical RAM is exhausted.

Symbolic link

A special file that simply points to the location of another file. Symbolic links are used to avoid data duplication when a file is needed in multiple locations.

Tagfile

A file used by the Slackware **setup** program during installation, which describes a set of packages to be installed.

Terminal

A human-computer interface consisting of at least a screen (or virtual screen) and some method of input (almost always at least a keyboard).

Toolkit, GUI

A GUI toolkit is a collection of libraries that provide a programmer with code to draw widgets such as scrollbars, checkboxes, etc. and construct a graphical interface. The GUI toolkit used by a program often defines its look and feel.

UID

User Identifier. A unique number that identifies a user to the system. UIDs are used by most programs instead of usernames because a number is easier to deal with; usernames are generally only used when the user has to see things happen.

VESA

Video Electronics Standards Association. The term VESA is often used to denote a standard specified by said Association. Nearly all modern video adapters are VESA-compliant.

Virtual terminal

The use of software to simulate multiple terminals while using only a single set of input/output devices (keyboard, monitor, mouse). Special keystrokes switch between virtual terminals at a single physical terminal.

Window manager

An X program whose purpose is to provide a graphical interface beyond the simple rectangle-drawing of the X Window System. Window managers generally provide titlebars, menus for running programs, etc.

Working directory

The directory in which a program considers itself to be while running.

Wrapper program

A program whose sole purpose is to run other programs, but change their behavior in some way by altering their environments or filtering their input.

X server

The program in the X Window System which interfaces with graphics hardware and handles the actual running of X programs.

X Window System

Network-oriented graphical interface system used on most Unix-like operating systems, including Linux.

[Prev](#)

Summary

[Home](#)

[Next](#)

The GNU General Public
License