Linux Guidebook

Joseph Colton

Version 1.0

Preface

Welcome to the Linux Guidebook. This book is intended to be a resource that you can use to help you better understand Linux and the abilities that it provides. This book was written while Red Hat Linux 7.1 was the popular one to use, but because the operating system is truly object oriented in nature, the things you learn by using this book will still be beneficial to you in the later versions to come. Linux is may favorite operating system and I hope that by reading and using this book you will also be able to come to love the Linux that I love.

Who should read this?

This book is intended for people who already know a little bit about computers and operating systems. In this book I plan to take the reader from the level of just basically understanding the shell commands to the level where the reader will have the resources to setup, and administer Linux on everything from a single computer to a small sized network. If you are planning on installing Linux or have a network to run and would like to know about Linux, this book is for you.

Reading this book

This book starts with the basics in the beginning of Part I and then goes on to talk about servers in Part II. Depending on what you are reading this book for, you should read different sections. If you would like to know everything about Linux then I recommend you read the whole book. Even after reading the whole book there will be many things that you can still learn. I hope that this book will put you on the right track and help guide you through your learning experience. If you would like to run Linux on your desktop then skip ahead to Section 1.2 and read the chapters recommended there. For servers skip ahead to Section 1.3. It is strongly recommended that you have a basic understanding of Networking and know a little bit about programming before you start this book. You will still be able to get a lot out of this book without this knowledge, but it does add to your learning experience.

In this book there are two prompts commonly used. Both are bash shell prompts. If the word bash is followed by the $ symbol then it means that any user should be able to type the command. If the word bash is followed by the # symbol then you must have root privileges in order to use the command. With root privileges I will also assume that you are logged in as root. These prompts will appear as in the following:

```
bash$ ls -al
```

and

```
bash# setup
```

If a user has a regular bash$ shell and switches to become a superuser that does not always set the path to be the same as the root path. The following is not exactly the same as logging in as root:

```
bash$ su root
```

If commands that you want to use are in the **/sbin** or **/usr/sbin** directories then you will have to type the directory before the command like this:

```
bash# /sbin/ifconfig
```

I will not usually type the **/sbin** or **/usr/sbin** suffixes, so you will have to remember to add them when they are necessary because you switched to root and your path has not changed.

All of the file names, special key combinations, URLs and commands will appear in the text as follows:

Example filename: **/proc/cpuinfo**
Example key combination: **Ctrl-Alt-Backspace**
Example URL: **http://www.cs.byuh.edu/research/colton/**
Example command: **ps aux**

Contacting the author

If you notice anything that needs correction or is unclear or if you would like to contact me for any other reason, feel free to email me at either **joseph@cs.byuh.edu** or at **joseph@colton.byuh.edu**.

Copyright

Contents

Part I

Learning Linux

Chapter 1

Introduction to the Linux Guidebook

1.1 What is Linux?

Many people have wondered the question "What is Linux?" The first thing people think is that Linux is a company. Because there is no company called Linux, people are often confused. The next thing people tend to thing is that Linux is a software program you install on another operating system. The software idea is closer, but not quite right. I will attempt to give you a little bit of information in the sections about the history, distributions, and about Open Source Software to help you decide that for yourself what Linux really is.

1.1.1 Linux History

Although the Linux Kernel was written by Linus Torvalds in 1991, what we know as Linux today started much further back. In 1968 the Internet (then known as ARPANET) only included four universities: The University of California at Los Angeles, SRI (in Stanford), the University of California at Santa Barbara, and University of Utah. Okay, lets skip ahead a little bit. ARPA became DARPA and later the Internet. Computing was limited to large computers until the the first home computers started coming out in the late 1970s. Part of the problem was that UNIX the common operating system cost too much money to run on the smaller computers. Home users had to settle for the only operating system in their price range. By 1990 the Internet had expanded to more than 300,000 host computers. A professor named Andrew S. Tanenbaum wrote from scratch the MINUX operating system and published a book called "Operating System". With this book was the code for the MINUX system. Computer Science students all over read the secrets of operating systems. Linus Torvalds, a student of Computer Science at University of Helsinki, inspired and helped by the GNU project and free software wrote the Linux kernel and released it as Open Source. With Linux an open source project, hackers all over the Internet community did their best to help test and write parts of the operating system. Today you can get Linux installed for almost every processor you can buy. Linux runs on everything from supercomputers to Palm Pilots.

1.1.2 Linux Distributions

Since Linux is free and most of the software for Linux is free there has to be someone to put the operating system and the software packages together. Red Hat is currently the most popular company that makes a Linux distribution, but there are many more. Some of the more common ones are Red Hat, Caldera, Debian, Mandrake, SuSE, Beowulf, Peanut Linux. Many of these distributions can be found on more than one processor chip. Supported chips include: the Intel 86 line, Sparc, MIPS, Alpha, PPC, m68k, etc. While there are many distributions, the one thing that they have in common is that they all use a Linux kernel. Basically the only real difference between the distributions is the software that comes with Linux and the default settings. You can find many of the popular distributions at **http://www.cheapbytes.com/**.

1.1.3 Open Source Software

Open Source software is similar to free software in that you can get it for free, but there is much, much more involved. Free software can come in an already compiled, ready to run form. Free software is available everywhere on the Internet. Open Source is free, but it allows you to see how the software was written and legally gives you rights to change and use that software source code to make something newer and better. Open Source is all about giving and friendship. In the Open Source Community, it is not how rich or powerful that makes you popular, it is how big of a gift you give to the community that makes you famous. While closed free software is nice to use, free open software encourages the community around you to grow and become better.

1.2 Using Linux as a Desktop

Linux is an operating system capable of running about any type of machine you would want. Linux can run on common desktops, but they are still missing some of the easy to use features that are found on some of the commercial operating systems. If you intend to run Linux as your desktop operating system then I recommend you read the following chapters:

Chapter 3 - Installing Red Hat Linux
Chapter 4 - Linux Shell Environment
Chapter 6 - Software Installation
Chapter 7 - Administration Tools
Chapter 8 - Data Backup

If you are planning anything more than just a stand alone machine then I would also recommend that you read whichever chapters seem to look interesting to you.

1.3 Using Linux as a Server

Linux is well known for its superior performance in the server world. If you are considering using Linux as your server operating system then I recommend you read the following chapters even if you do not think that you will run all of the

servers:

Chapter 3 - Installing Red Hat Linux
Chapter 4 - Linux Shell Environment
Chapter 6 - Software Installation
Chapter 7 - Administration Tools
Chapter 8 - Data Backup
Chapter 9 - Security
Chapter 11 - MySQL Server
Chapter 12 - Web Server
Chapter 13 - Mail Server
Chapter 14 - NFS/NIS Servers
Chapter 15 - DNS Server
Chapter 16 - DHCP Server

1.4 Preparing for Disaster Recovery

In organizations the janitors have keys everywhere and the presidents have money. System Administrators usually get keys, a cut of the money, and passwords to the servers. From this it looks like the administrators have everything. There are reasons for this. With the keys you can come in anytime to fix things. With passwords you can work anywhere, and with the cut of money you can order the parts to give yourself more work. In this chapter we will cover some of the activities that system administrators might have to do in Linux. I will also give you an overview of the rest of the book and how it ties in with Linux administration.

1.4.1 Administration Knowledge

Knowledge is the key to being a good Administrator. Any administrator can figure out how to do something, but it is the ones who already have an idea about what to do that are the quickest and considered the best. Throughout this book you will look at different aspects of Linux administration. In the following paragraphs we will look at some of the following chapters and see how they tie into the job of the System Administrator.

Networking Administrators need to understand how the network is working. What are the available IP addresses and where are all of the services running? How can the network be build to work smoothly? What do all of the terms mean? Not only do you need to understand how it works, but you need to know how to build or setup a network.

In Chapter 2 we will review some of the major term and give a little bit of information about each one. It is strongly recommended that you have a basic understanding of networking. If you are still struggling with networking terms and concepts, look over the networking chapter and lookup the subjects you do not understand.

Installation Installation is the base of the setting up and managing computers. Through installation you come to understand how it is setup and how you

can make the network better. It is choose things such as partitioning so that you can prevent Denial of Service attacks, or have a way to reinstall without losing personal data. Chapter 3 will cover some of the basics of the installation process and give you further insight into what you are doing.

The Shell The Shell is the system administrators best friend. Unfortunately, it is also often the regular users worst enemy. As system administrator it is important to not only know the shell, but also to use it so that you are better able to manage the system. The shell enables many things that are not possible in the ever changing world of the GUI. The shell stays fairly constant and is extremely powerful. Sometimes in a system recovery the shell is all that you have to work with.

Chapter 4 will go into some of the many possibilities available with the aid of the shell.

Software Packages In any organization with more than one user there will be times when there will be a need (or at least a desire) for some software package that is not currently installed on the system. In Chapter 6 we will cover some of the many software file formats available on the Internet and the basics of how they are commonly installed.

Administration Tools Administrators need to have tools which they can use to get a given task done. Sometimes you want to start servers, create accounts, or fix something. In Chapter 7 we will look at some of those tools and how to use them.

System Backup System backups can be vital in a changing network environment. Most places do a system backup at least once a week. Backing up the system can save the system administrator a lot of grief and suffering when disasters strike. In Chapter 8 you will learn about a few different ways you can go about doing a system backup.

Security In security, knowledge (or knowing) is usually the most important thing. System administrators need to be aware of everything that happens on their system. This requires checking the logs and doing frequent checks on the settings you have set. It possible that things can be changed on the system without anyone ever noticing. I have heard it said that the average computer gets port scanned 15 minutes after coming online. Sometimes the first you know about you system being hacked into is when your system starts to attack someone else and they let you know. This can be quite a hit on your personal pride. Chapter 9 will go into this subject.

Servers Servers are important if you if you have any interaction between computers you are in charge of. You need to be aware of the available servers and know the risks and advantages of running each one. In Part II you will learn about some of the major servers and how to get them setup and running.

1.4.2 Disasters

What does it take to have a disaster? What is a disaster? These are hard terms to describe. You can take the extremes and compare. If everything is working fine then there is no disaster. If the building catches on fire and the computers all burn up then that is a disaster. I guess we will define a disaster as something that the system administrator can do that other people cannot.

Forgotten Passwords Forgotten Passwords has got to be the most common problem around. People forget their passwords and then come to the system administrator. In Linux you can reset someones password by typing some thing like this:

```
bash# passwd bob
Changing password for user bob
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

This is one that you will just have to get used to. People are not the best at memorization.

Deleted Files Have you ever deleted a file and asked a system administrator to help you get it back. In Linux this can be difficult. Sometimes the file can be recovered, but sometimes it is gone. If you do a good job at backing the system up you will probably have a recent copy of the directory. Unfortunately, people tend to lose files they are working on. Sometimes the files are not in the backup or they have been modified signifigantly since the last backup.

Programs not working Programs usually do not work because they were either written poorly, they were written for a different version of the operating system or libraries, or they have not been installed properly. Installation is something that we want make sure you have a better chance to do right. Chapter 6 focuses on the software installation process and helps you do is better.

Hardware failures Hardware failures are a bit out of the scope of this book, but this book does have a chapter of data backup. If you would like to be prepared for hardware failure, then Chapter 8 may be the one for you.

Viruses There have been rumors of Linux viruses, but as of yet I still have never seen a Linux virus. I guess the virus programmers need to work harder. If you are working with a Windows machine and have a Linux server then you might need to do something about viruses, but only because of the processor time and space that can be consumed by virus infected machines. Viruses are out of the scope of this book.

Hackers/Crackers System breakins are probably the worst problem. You do not have to be doing anything bad to get your machine compromised, sometimes it is out of your control. The best you can sometimes do is to protect the system

the best you can and backup frequently. Chapter 9 covers a few security issues and gives you information to get started down the "secure" trail.

Chapter 2

Networking Basics

2.1 Names and Addresses

The hardware address, IP address, and hostname are names that computers on a network can use to talk to each other.

2.1.1 Hardware or MAC Addresses

Every network interface has a hardware address. These interfaces include network cards, network probing equipment and other things like advanced switches. These addresses are 6 bytes long and are usually written in two character hex sets separated by colons (01:23:45:67:89:ab) or dashes (01-23-45-67-89-ab). These addresses are used so that two computers next to each other can talk to each other.

2.1.2 IP Addresses

In order for computers to be able to talk to other computers on the Internet, they need to have addresses. The Internet Protocol (IP) addresses were setup in such a way that information could find its way to a destination IP address. The numbers were not randomly given out. They were given out in groups, then subdivided into smaller groups until they were eventually allocated to individual computers. With IPv4 IP addresses are usually written as four positive numbers separated by a ".". Each of these numbers is made up of 8 bits. The smallest a number can be is 0 and the largest it can be is 255. A small example number would look something like 1.2.3.4 and a large number would look something like 250.251.252.253. Since each of the four numbers is made up of 8 bits, the whole IP address is made up of 32 bits. With 32 bits there are a total of 4,294,967,296 possible IP addresses.

2.1.3 Hostnames

IP addresses are not very easy to remember. If you had the choice between remembering your friends by their names or by their telephone numbers, which one would you choose? Telephone numbers are divided into sections that help identify where you are, but names give you a description. Names are much more

friendly and they are easier to remember. Hostnames have the same purpose as real names. They are arranged with a host and a domain. If my host computer was named **red** and the domain was **paintcolors.com** then the hostname would be **red.paintcolors.com**. With a host name you can remember the site much better than if you had to remember the IP address. Like names and telephones, there is a way to look up the IP address of a computer if you have the hostname. Later in this chapter we will talk about Domain Name Servers (DNS) the phone books of the Internet. A problem with hostnames is that you cannot really share the same name as some other computer since the whole world can use the same name to talk to you.

2.1.4 Domain Names

The domain name of a hostname is what you would get if you took the front word and dot off. If you had a hostname of **www.google.com** then the domain name would be **google.com**. It gets deeper still. The domain name of **google.com** is **com**. And the domain name of **com** is **.** Okay, maybe that is too deep, but you probably get the idea.

2.2 Gateways and Routers

2.2.1 Local Area Network (LAN)

In understanding the purpose of a gateway or router it is necessary to understand the idea of a Local Area Network (LAN). Computers that can talk to each other without talking to another computer first are in the same LAN. If you have two computers in the same room, chances are good that they are probably in the same LAN. Computers on the same network are either directly connected together with a crossover cable, or are both connected somehow to the same HUB or switch.

2.2.2 HUBs and Switches

A HUB is a device with multiple network ports that listens to network traffic on each port and echos that same traffic back out on all of the rest of the ports. In a sense a HUB works like an amplifier. A switch is a little bit different. The basic idea is the same, but the switch remembers which computers are connected to each port. When a switch hears the network traffic on port A it looks to see where that traffic is bound. If the traffic is bound for a computer on port B, then the switch only echos the traffic down port B. No other ports hear the communication between the computers on port A and port B. Switches can also be placed between two networks. In this case the switch would usually have two IP addresses and would direct traffic from one network to another one.

2.2.3 Gateway

A gateway is an entrance point from one network into another network. The device has an IP address that is designated as a gateway address to your network. If you want to send Internet traffic from you network to another network you send it to the gateway and expect it to get to where it is supposed to go. The

gateway device only has to provide one gateway address in order to be considered a gateway, but needs to be connected to both networks.

2.2.4 Routers

Routers are any machine that listens to traffic and on one network, then directs it to go somewhere on another network. With this definition, HUBs would would not really fall under the definition of a router, but a switch might. A gateway is a router because it routes network traffic from one network to another. Routers can be a part of more than two networks. Routers can act as the gateway on many different networks all at the same time. Sometimes there can be more than one router connected to the same networks. With this kind of setup you can send network packets out one gateway address and receive the in coming packets from an entirely different machine and address.

2.3 Domain Name Servers

We mentioned before about how hard it is to remember a machines IP address. In order to be able to forget the IP address we use hostnames instead. The Domain Name Servers is like a phone book where you can look up IP addresses by giving the hostname. You can use the **nslookup** and **dig** commands to look-up IP addresses from the shell. Here are some examples:

```
bash$ nslookup www.osdn.com -silent
Server: 64.75.142.2
Address: 64.75.142.2#53

Non-authoritative answer:
www.osdn.com canonical name = osdn.com.
Name: osdn.com
Address: 64.28.67.20
```

With **nslookup** you get back the DNS server you used and the port that server was at. You then find out that **www.osdn.com** is just an alias to the real name of **osdn.com**. The name **osdn.com** points to the IP address **64.28.67.20**. So you found the IP address. Now let's try **dig**.

```
bash$ dig www.osdn.com

; <<>> DiG 9.1.0 <<>> www.osdn.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12056
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.osdn.com. IN A

;; ANSWER SECTION:
www.osdn.com. 17709 IN CNAME osdn.com.
osdn.com. 17709 IN A 64.28.67.20
```

```
;; AUTHORITY SECTION:
osdn.com. 17709 IN NS ns2.andover.net.
osdn.com. 17709 IN NS ns1.andover.net.

;; ADDITIONAL SECTION:
ns1.andover.net. 104109 IN A 64.28.67.55
ns2.andover.net. 104109 IN A 209.192.217.105

;; Query time: 24 msec
;; SERVER: 64.75.142.2#53(64.75.142.2)
;; WHEN: Tue Feb 26 10:30:50 2002
;; MSG SIZE  rcvd: 139
```

The **dig** command gives back more information than most people can easily understand. One way to start is to skip all of the lines that start with a semicolon. The semicolon lines are not very important. The data that comes back looks much like the format of a DNS configuration file. The first uncommented line is the one that says that **www.osdn.com** is an alias to **osdn.com**. The next line says that **osdn.com** points to **64.28.67.20**. The next four uncommented lines tell you which DNS to talk to to get current correct information.

2.3.1 CNAME

The CNAME or canonical name of a machine is the real name. A machine should only have one canonical name. The other names then become aliases that point to the CNAME or real name. The canonical name then points to the IP address that the machines has.

2.4 Network Addresses, Masks, and Broadcast

In any Local Area Network there is a range of addresses that you can claim as part of the network. Without this range of addresses you would never be able to figure out if a network packet was supposed to be delivered locally to a machine that was part of the LAN or if the packet needed to be sent to the gateway for delivery on a network somewhere else.

2.4.1 Network Address

In every network there is a network address. The network address is the lowest IP address that is part of the network. If I had a network that had all of the addresses from **1.2.3.0** to **1.2.3.255** then the network address for that network would be **1.2.3.0**. There would be 256 IP addresses that belonged to the network. Every network address ends in an even number.

2.4.2 Calculating the Network Mask

If you know how many addresses are in your network then you can calculate the network address very easily. Lets first assume that you have 256 addresses or less. If this is the case then the network mask will start as **255.255.255.**

something. You can take the number 256 and subtract the number of addresses that you have in your network to give you the last number in your mask. Here is a simple table:

IP Addresses	Network Mask
256	255.255.255.0
128	255.255.255.128
64	255.255.255.192
32	255.255.255.224
16	255.255.255.240
8	255.255.255.248
4	255.255.255.252

You can take those numbers and convert them into bits and count the bits. The number 255 converted to binary is 11111111. That is 8 bits. The whole IP address is 32 bits broken into groups of 8 bits. 255.255.255.0 would be converted to 11111111.11111111.11111111.00000000. If you look at the number you will notice that the number is a group of 1s followed by a group of 0s. We would call this a /24 network because there are 24 1s. All of the network masks can be converted into a group of 1s followed by a group of 0s. 255.255.255.128 would be converted into 11111111.11111111.11111111.10000000. This would be a /25 network because there are 25 1s in a row before the 0s.

2.4.3 How to use the Netmask

Once you can figure out what the Netmask of a network is you are probably wondering how to use it. All IP addresses get translated from their four groups into one large number before they are sent across the Internet. Once they are in this format the Linux Operating System looks at the number to figure out where to send the network packet. If your computer is 1.2.3.4 and you are trying to send a packet to 1.2.3.5 and your network mask is 255.255.255.0 or a /24 network then you would first translate the addresses into a group of bits. 1.2.3.4 would become 00000001.00000010.00000011.00000100 and 1.2.3.5 would become 00000001.00000010.00000011.00000101. If you line them up better you can see which parts are the same. Lets mark a 1 below each bit that is the same until we get a different bit. When we get a different bit we will fill in the rest with 0s:

1.2.3.4	00000001.00000010.00000011.00000100
1.2.3.5	00000001.00000010.00000011.00000101
same bits	11111111.11111111.11111111.11111110

We have 31 bits that are the same. In order to be in the same network we need to have at least 24 bits the same. That means that this address is in the same network and can be sent to another computer locally without being sent to a gateway. What if the same address 1.2.3.4 was sending a packet to 1.2.34.56? What would the table look like then?

1.2.3.4	00000001.00000010.00000011.00000100
1.2.34.56	00000001.00000010.00100010.00111000
same bits	11111111.11111111.11000000.00000000

In this example we only have 18 bits that are the same. We needed 24 bits to be the same in order to be in the same network, so we will send the packet to the gateway. Pretty easy? Well, it takes a little bit of time for some people to understand this concept.

2.4.4 Broadcast Address

The Broadcast Address the last addresses in the network. If you send packets to this address the packets are supposed to be sent to every address in your LAN. Since there was a lot of confusion when the Broadcast address was invented people can sometimes send packets to the Network Address and get them to Broadcast to all of the machines on the network.

2.5 Trouble-shooting a Network Computer

When you are trying to find and solve a network problem can be a very difficult task. Fortunately there are tools that can help you figure out why things are not working as they should.

2.5.1 Do you have an IP address?

I find that the lack of an IP address is one of the best indications of why you cannot see or communicate with the network. If you do not have an IP address there is a follow-up question you can ask. Am I using DHCP? If you are using DHCP and do not have an address, there are a few things that that could mean. Either you are not connected to the network, the DHCP server is not connected to the network, or the DHCP server is not giving you an address. To fix the problem, make sure your computer is connected to the network and run the following command:

```
bash# /etc/rc.d/init.d/network restart
Shutting down interface eth0:                              [  OK  ]
Setting network parameters:                                [  OK  ]
Bringing up interface lo:                                  [  OK  ]
Bringing up interface eth0:                                [  OK  ]
```

If everything says OK then you should have an IP address. To see the IP address, type:

```
bash# ifconfig
eth0      Link encap:Ethernet  HWaddr 01:23:45:67:89:AB
          inet addr:1.2.3.4  Bcast:1.2.3.255  Mask:255.255.255.0
          IPX/Ethernet 802.2 addr:CD956500:00C0F0407A8B
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1920018 errors:0 dropped:0 overruns:0 frame:0
          TX packets:349874 errors:5 dropped:0 overruns:0 carrier:5
          collisions:0 txqueuelen:100
          Interrupt:11 Base address:0x1000
```

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:23510 errors:0 dropped:0 overruns:0 frame:0
          TX packets:23510 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

In this example the IP address is 1.2.3.4. If you still do not have an IP address you can try to configure a static IP address using **linuxconf**, **netconf**, **netcfg**, or **netconfig**.

2.5.2 Can you talk to your Gateway?

You need to know what the IP address of your gateway is before you can test to see if you can talk to it. Once you know the IP address, see if you can **ping** the gateway:

```
bash$ ping 1.2.3.1 -c 3
PING 1.2.3.1 (1.2.3.1) from 1.2.3.4 : 56(84) bytes of data.
Warning: time of day goes back, taking countermeasures.
64 bytes from 1.2.3.1: icmp_seq=0 ttl=255 time=5.691 msec
64 bytes from 1.2.3.1: icmp_seq=1 ttl=255 time=493 usec
64 bytes from 1.2.3.1: icmp_seq=2 ttl=255 time=487 usec

--- 1.2.3.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.487/2.223/5.691/2.452 ms
```

If your gateway was 1.2.3.1 and you got results like the results above then that means that everything is working right. If you cannot **ping** the gateway you will get results that look a little bit like this.

```
bash$ ping 1.2.3.1 -c 3
PING 1.2.3.1 (1.2.3.1) from 1.2.3.4 : 56(84) bytes of data.
From 1.2.3.4: Destination Host Unreachable
From 1.2.3.4: Destination Host Unreachable
From 1.2.3.4: Destination Host Unreachable

--- 1.2.3.1 ping statistics ---
3 packets transmitted, 0 packets received, +3 errors, 100% packet loss
```

If the gateway is unreachable you need see if other computers on the same network can talk to the gateway. If they can talk to the gateway, but you cannot, see if you can talk to them. It is possible that you are not on the same network, or that your netmask is not the same as theirs. Everyone on the network should have the same network mask.

2.5.3 Can you talk to your DNS?

If you can talk to your gateway, but cannot seem to get out on the Internet, that could mean that you are not pointing to a DNS server correctly. Try looking up some names with your DNS server:

```
bash$ nslookup www.linuxnews.com -silent
Server: 1.2.3.2
Address: 1.2.3.2#53

Non-authoritative answer:
Name: www.linuxnews.com
Address: 207.247.9.92
```

If this command gives you basically the same results, then your DNS server is working fine and you are connected to it. If it does not work, then you might need to check the DNS servers to see it they are correct. You can check your DNS settings using the commands **linuxconf**, **netconf**, or **netcfg**. All of the Domain Name Servers you use are also listed in your **/etc/resolv.conf** file.

```
bash$ cat /etc/resolv.conf
domain cs.byuh.edu
search cs.byuh.edu
nameserver 1.2.3.2
nameserver 1.2.3.3
```

If this list does not look correct to you, or if there are no name servers listed, then that would be something that you need to take care of.

If you have an IP address, you can ping your gateway, you can get names back from the DNS, and still cannot talk to the outside world, then it is possible that the problem comes from your Internet Service Provider (ISP), there are two computers using the same IP address, or you might have some type of firewall problem.

2.5.4 A second look at the Gateway

Sometimes people get their IP and Gateway numbers backwards when they are setting up their computers. Do not ask me why? Just believe that this happens. In order to discover this problem Linux has a great server called **arpwatch** that watches the MAC addresses and IP addresses on your network line. You can run this program from root:

```
bash# arpwatch
bash# pine
```

The first command starts **arpwatch**, then the second one takes you into a mail reader. Press **i** to view your Inbox. Scroll to the bottom and wait for a few moments. When you think something might have happened press the **down arrow** a few times to see if any new messages are in you Inbox. Email messages should appear if you are on an active network. See if you have an email message with your gateway IP address. A message would look something like this:

```
Date: Thu, 14 Mar 2002 14:52:16 -1000
From: Arpwatch <arpwatch@bob.org>
To: root@bob.org
Subject: new station (dave.bob.org)

            hostname: dave.bob.org
```

```
      ip address: 1.2.3.8
ethernet address: 0:10:20:30:2:f2
 ethernet vendor: Dell
       timestamp: Thursday, March 14, 2002 14:52:16 -1000
```

You should get a email message with the gateway's ethernet address. If you later get a different ethernet address you know that there might be another computer using the same IP address. It is also possible that the gateway machine has more than one ethernet address on the local network.

Chapter 3

Installing Red Hat Linux

For the following instructions I will assume that you are planning to learn about Linux. We will be doing a complete installation that includes everything contained on the CDs. I am also going to assume that you have a computer with at least 10 gigs of hard drive space, 64+ megs of RAM, and a bootable CD-ROM drive.

3.1 Installation Overview

The installation process is one that gives the user many insights into how the computer works. As we go through the installation we will discuss the disk partitions, the networking options, firewall configuration, accounts, packages, settings and more. After the instructions in each section we will cover information about how to change the settings once the computer is setup and fully installed. I will also talk about the risks that are involved with incorrectly configuring your computer.

3.2 Starting the Install

Insert the first Red Hat Installation Disk in the CD-ROM drive and reboot the computer. The computer should boot up to a text screen that lists different CD options. From here you can either type: **linux** and press **Enter** or just press **Enter** and default to Linux. This will start the installation process.

The first question you will be asked is what language to install Linux in. Select English and press **Next**.

Next you will be asked your keyboard layout. Unless you specifically think your keyboard is different, select **Next**.

When asked about your mouse you need to make sure you know what type of mouse you have. A PS/2 mouse is a round connector on the back of your computer. Serial mice connectors are bigger and connect to a D-shaped socket with 9 pins that looks similar to your monitor connector. A USB mouse would fit in a basically flat thin rectangular hole.

If you change the default and put something other than your real mouse your X server might not work properly.

After selecting your mouse Press **Next** to install, then select Custom System and press **Next**.

3.3 Understanding the Disk Partitions

Red Hat Linux requires two and on later versions three partitions in order to install correctly. The easiest way to create the disk partitions for a beginner is with Disk Druid. You first need to delete any unwanted partitions. To do this select the partition and press the **Delete** button for each one. Read the information to the left. Here is a little bit of information about some suggested partitions.

```
Linux recommended partitions:

/      - This is the default directory everything is put into
/boot  - This directory contains the kernel
/home  - This directory contains the user files
/var   - This directory contains system logs and now web pages
/usr   - This directory contains programs everyone uses
/bin   - This is where basic commands are located
```

3.3.1 The /boot Partition

So what do we need and what do we want? We need a /boot directory. It only needs to be a few megabytes. I usually give 50 megabytes because I feel generous. You might be able to get away with 4 megs, but that is only for a few more years.

3.3.2 Swap Space

Next we need a swap partition. Usually it is good to create a swap partition twice as big as the amount of RAM that you have. If you have lots of RAM and cannot create a SWAP partition as big as twice your RAM, you can make a few smaller swap partitions.

3.3.3 The /var Partition

Once the swap partition is done you can go on with the rest of the partitions. A **/var** partition is good because if you have a network machine you will probably generate a lot of logs. If a cracker were to break into your computer, the cracker would probably run a program to erase the log files. If they are stored on a separate partition you have a higher chance of being able to recover those erased files. Also, there are Denial of Service (DoS) attacks that crash machines that do not have their logs on a separate partition. Keep the logs separate and you will have a better experience. 500 megs to 1 gig for var should do the job for most people.

3.3.4 The /home Partition

Home directories stored in the /home partition are desirable. If you think that you may re-install at a future time, but you do not think that you will want to backup all of the home directories and restore them, then the /home partition is for you. If you are the only user for your system you can have a small space for /home, but as more users need to use the system, you will need more space for the /home directory. It is good to make the /home partition as big as you can reasonably make it. On a 20 gig machine 10 gigs should be fine. About 10 gigs less than the total usually works good.

3.3.5 The / Partition

The last, but a very important partition is the / partition. The / partition contains the /root, /etc, and /sbin directories. That is why they cannot be made separately. The / partition needs to be big enough for all of these and for everything you did not already allocate. It is usually a good idea to make it at least 5 GB. Remember that the /root directory is included in this so everything that root wants to do in secret will have to be done in this space.

3.3.6 RAIDs

A RAID is a collection of drives that work as one drive. To create a RAID first create RAID partitions that are the same size, then create the RAID using the **Make RAID** device button. Select the RAID partitions to use in the RAID device, and select the partition type to be Linux Native, ext2, or ext3. Set the device to md0 or md1 if you already have a md0 RAID device. Set the mount point then you click **OK**.

3.3.7 Format

Once you have all of your partitions you can continue on with your installation. Press **Next** and wait for a screen that lets you select which partitions to format. If you have an old partition that you are using, but you did not change the size of it then it should be unselected. New partitions should all be selected. Decide which ones to format, then press **Next**.

3.4 Networking Options

If you are setting your computer up to be able to run servers you need a static IP address. If it is just a workstation then you can usually do everything getting an IP address from the DHCP server if you have one. If you have a DHCP server still unselect DHCP and fill in you hostname then reselect DHCP. Here is where you can cause a lot of problems for the network. If you get the IP address and the gateway address mixed up, you will probably mess up more than just your computer. Make sure you get the addresses correct.

3.5 Firewall Configuration

People are often confused into believing that a firewall is always something good. That is not always the case. Firewalls protect you from the Internet in much the same way that disconnecting you computer from the Internet would. The main difference between the two is that if you are disconnected you cannot use the Internet. If you have a firewall then the you can only use parts of the Internet. A misconfigured firewall is as bad as not being connected. A skillfully configured firewall can do great things. For the installation process it is better to select no firewall and then configure it later. You can configure firewalls in text mode with **ipchains** and **iptables** or you can use GUI programs to use these tools. Firewall GUIs include **firewall-config** and **lokkit** (which can also be run by typing **setup**).

3.6 Accounts and Authentication

You have to decide how to authenticate to your machine. If you are running just a single machine you would not want to use any of the server methods to login to your machine. If you are using a network full of clients then authenticating with a server might make sense. The server authentication methods are Network Information Server (NIS), LDAP, Kerberos, or Hesiod. If you want to use any of these you will have to select the server authentication method and enter in the server and other information. NIS will be covered in Chapter 14. If you are not using an authentication server then the default options should be good. Just press **Next**.

3.7 Packages

If you are in this to learn Linux you should probably install everything. You can select all of the packages individually, or you can scroll down to the bottom and select the package entitled "Everything." If you want to select the packages individually you can pick from a long list that includes Servers, Development (compilers), Text editing, Kernel Development, and many more. When you first begin it is probably to install everything and learn what is useful. You can later decide what you do and do not like.

3.8 Graphical Settings

Linux does a good job of detecting your graphics card and monitors most of the time, but sometimes it does not do a great job. If it looks like Linux got everything right then you can just press **Next** until you get to a screen where you can test your settings. Make sure that you test your settings before you continue. If nothing seems to be working correctly then you might want to choose Text as your login type and continue on. You can later change your graphical settings using the **Xconfigurator** command from the shell as root. If you are having a hard time spelling **Xconfigurator** you can use the command **setup** and select X configuration.

3.9 Kickstart

In Linux you have a great way to install a computer without being at the computer to answer all of the questions. With the program called Kickstart you make the installation decisions ahead of time and store the answers to the installations questions in a file called **ks.cfg**. There is a GUI program called **ksconfig** that you can use to create this **ks.cfg** file. When you have this file you can put it on a boot disk created with the **dd** command. All you need to do is get a boot image from the **images/** directory on the Red Hat Linux cdrom. To put the image on a floppy run this command:

```
bash# dd if=bootnet.img of=/dev/fd0
```

There are other images, but the **bootnet.img** gives you the ability to do network installations. Next copy the kickstart file onto the disk:

```
bash# mount /dev/fd0 /mnt/floppy/
bash# cp ks.cfg /mnt/floppy/
bash# umount /mnt/floppy/
```

With this done you can reboot the machine. At the boot: prompt you can type:

```
boot: linux ks=floppy
```

Kickstart should take over the installation and only ask questions that are not answered in the kickstart file.

3.10 Network Installations

You can easily do a network installation from a floppy disk. You first need to get a good floppy disk. In order to to the network boot the disk needs to have only good sectors on it. Next you need to find the boot image on the install device. If you have the Red Hat installation CDs you can go to the images directory on the first CD and then find the **bootnet.img** file. In Linux you can copy this file to a floppy using the following command:

```
bash# dd if=bootnet.img of=/dev/fd0
2880+0 records in
2880+0 records out
```

The the file **bootnet.img** contains the format information and everything else, so you do not need to worry about formatting the disk first. Once you have copied this file to the disk everything that was there before is gone. There is no real way to recover the data that was there before. If you get any output that looks a little bit different then it is possible that you have a bad floppy disk. Output for a bad floppy would look something like this:

```
bash# dd if=bootnet.img of=/dev/fd0
dd: writing to '/dev/fd0': Input/output error
2289+0 records in
2288+0 records out
```

If you get this kind of output then you need to look for another disk to do the job. It should be pretty easy to find a good disk if all of them are new, but even some of the new disks are not good enough for an install.

After the disks you need to decide what type of network install you are going to provide. It is possible to offer all three types (ftp, http, nfs) from the same server with the same set of files, but some of these services are easier to setup than others. With ftp you have to allow connections to your ftp port. That is not always safe, but it is fairly easy to do. With NFS you have to export the file partition that contains the install directory tree, but you have to specify which machines to export to. NFS is a fast way to do a network install, but sometimes it takes a long time to get things working correctly. HTTP is one of the easiest and most secure ways to do a network install. In order to do a HTTP installation you need to copy the files from each of the CDs into a directory on the HTTP server. If you were going to put the install image in a directory called **/var/www/html/cdrom/** you could use the following command if the CD were mounted:

```
bash# cp /mnt/cdrom/ /var/www/html/ -r
```

You would have to mount each CD, run the command, then unmount the CD in order to get all of the installation packages in a subdirectory. If you do not have the CDs but you have the CD images instead you could use the create a temp directory called **cdrom/** and then mount the image instead:

```
bash# mkdir cdrom/
bash# mount -t iso9660 -o ro,loop=/dev/loop0 cdromimagefile.iso cdrom/
bash# cp cdrom/ /var/www/html/ -r
```

You would then have to mount, copy, and unmount the images instead of the CDs.

3.11 Dual Boot Machines

It seems like every operating system wants to be in the master boot record (MBR). Even Linux likes to be there. This is because the Master Boot Record is the first place that the BIOS goes to start loading your operating system. With a single boot machine this does not really matter, but on a dual boot machine this becomes important. Figure out which operating systems you want to install. Once that is figured out you will need to figure out which one you want to boot the other operating systems. Linux is good at booting operating systems that are in FAT partitions, but still does not do well with NTFS and some others. The operating system that boots the others should be installed last so that it is installed in the master boot record. In the **lilo** configuration section of Linux installation you will want to type in boot labels for each OS that you want to boot. Then you will need to select the OS that you want booted by default. After installation you can modify the default OS by editing your **/etc/lilo.conf** file and changing the default label to boot.

Chapter 4

Linux Shell Environment

4.1 Directory or Folder Navigation

Navigation with the shell is an extremely important skill for anyone who plans to use their computer for more than document processing and multimedia. Every operating system has a underlying file structure or way of storing files and data. Once you understand how everything is setup you will be able to do lots more with your computer.

4.1.1 Where am I?

This is the first question you should ask if you want to know a lot about your system and how to use it. Fortunately there is a command that can tell you what you need to know. As soon as you get a terminal window open you can type the command:

```
bash$ pwd
/home/bob
```

This should respond back with the path your current working directory. The **pwd** command stands for Print Working Directory.

4.1.2 Where can I go?

In the shell type the command:

```
bash$ ls -al
total 48
drwx------    4 bob      bob          4096 Feb 26 10:33 .
drwxr-xr-x    5 root     root         4096 Feb 26 10:33 ..
-rw-r--r--    1 bob      bob            24 Feb 26 10:33 .bash_logout
-rw-r--r--    1 bob      bob           224 Feb 26 10:33 .bash_profile
-rw-r--r--    1 bob      bob           124 Feb 26 10:33 .bashrc
-rw-r--r--    1 bob      bob          5450 Feb 26 10:33 .canna
drwxr-xr-x    2 bob      bob          4096 Feb 26 10:33 Desktop
-rw-r--r--    1 bob      bob           747 Feb 26 10:33 .emacs
```

```
drwxr-xr-x    3 bob      bob          4096 Feb 26 10:33 .kde
-rw-r--r--    1 bob      bob          3728 Feb 26 10:33 .screenrc
-r--r--r--    1 bob      bob          1019 Feb 26 10:33 .wl
```

This should list all of the files in the current directory. If you look at the output you will see a list of letters and dashes, a number, two names, a number, a date then a file name for each file. Each entry that starts with a **d** is a directory. You can change into any of those directories as long as you have permissions.

4.1.3 Changing Directories

You can change directories using the **cd** command. Here is a table of the **cd** command, the starting directory and the directory after the command for the user Bob:

Command	Starting Directory	Ending Directory
bash$ cd .	/tmp	/tmp
bash$ cd ..	/tmp	/
bash$ cd /home/bob	/tmp	/home/bob
bash$ cd	/tmp	/home/bob
bash$ cd ../home/bob	/tmp	/home/bob
bash$ cd ../../tmp	/home/bob	/tmp
bash$ cd /tmp	/home/bob	/tmp
bash$ cd bob/	/home	/home/bob

There are an endless number of ways to type in where you want to go. The only real problem is figuring out where you want to go.

4.2 Command Line Options

With Red Hat Linux the standard default shell is the **bash** shell. The **bash** shell offers many options at the command line. Here are some of the following options.

4.2.1 Simple Commands

The simple command is one that you can type on a command line all by it self. An example of a simple command line option would be to type:

```
bash$ whoami
bob
```

The command should then respond with information about who the current user is.

4.2.2 Arguments

You have probably typed in many commands with arguments. The arguments are what appears on the command line after the command. Here is an example of a command with an argument:

```
bash$ which whoami
/usr/bin/whoami
```

In this example the command is **which** and the argument is **whoami**. This will respond by telling you where the **whoami** command is located. If you have two commands with the same name you can use the which command to figure out which one will run if you type the command.

4.2.3 Redirecting input and output

Redirection of input and output is very important to system administration. Let's say that you want a list of every file on your computer, but you want to store that information in another file. You can redirect the output with the greater than symbol. Type in the following command:

```
bash$ find / -print > all.files
```

This will generate a files called all.files that will contain a line for each file that gives the complete path name of the file. You can also redirect input in much the same way. Input is redirected with the less than symbol.

4.2.4 Running in the Background

You can run commands in the background by typing an ampersand symbol after the command like this:

```
bash$ netscape &
```

When you are in graphics mode this will open **netscape** in a new window and also give you control of your shell. If you just type **netscape** your shell will be locked up and it is not always easy to get it back. In some programs you can press **Ctrl-Z** to get the program to stop and regain control of the shell. Once you have the shell you can type **bg** to get the program to continue running in the background. That is basically equivalent to starting the program in the background to start with.

4.3 Listing Files and Directories

Listing the files is a directory is not very complex. You can type the well known "**dir**" command or you can use the UNIX **ls** command. The **ls** command is short for list. To list all of the files in the directory in a long format type:

```
bash$ ls -al
total 48
drwx------    4 bob      bob          4096 Feb 26 10:33 .
drwxr-xr-x    5 root     root         4096 Feb 26 10:33 ..
-rw-r--r--    1 bob      bob            24 Feb 26 10:33 .bash_logout
-rw-r--r--    1 bob      bob           224 Feb 26 10:33 .bash_profile
-rw-r--r--    1 bob      bob           124 Feb 26 10:33 .bashrc
-rw-r--r--    1 bob      bob          5450 Feb 26 10:33 .canna
```

```
drwxr-xr-x    2 bob      bob          4096 Feb 26 10:33 Desktop
-rw-r--r--    1 bob      bob           747 Feb 26 10:33 .emacs
drwxr-xr-x    3 bob      bob          4096 Feb 26 10:33 .kde
-rw-r--r--    1 bob      bob          3728 Feb 26 10:33 .screenrc
-r--r--r--    1 bob      bob          1019 Feb 26 10:33 .wl
```

If you wanted to see the same files in a lot less space you could type:

```
bash$ ls -a
.       ..   .bash_logout  .bash_profile  .bashrc  .canna  Desktop
.emacs .kde  .screenrc     .wl
```

The files that have a leading period are the hidden files. If you were to leave out the **-a** argument the list would look like this:

```
bash$ ls
Desktop
```

Not a very long list. Since **Desktop** is a directory you can see the contents of that directory by typing the directory after the **ls** command:

```
bash$ ls Desktop
Autostart       kontrol-panel  Linux Documentation  Printer
www.redhat.com
```

4.4 Managing Files and Directories

When you get comfortable with your shell interface, managing files becomes easy. There are just a few basic commands that you need to know in order to manipulate the files the way that you want to.

4.4.1 Creating Files and Directories

Creating files is easy. All you have to do to create an empty file is use the **touch** command. Let's create a few files:

```
bash$ touch newfile.txt
bash$ touch newfile2.txt
bash$ touch newfile3.txt
bash$ touch newfile4.txt
bash$ touch newfile5.txt
```

This will create five new files, the first being called **newfile.txt**. If the file is already there this command will change the modification date to the current date. This can be very useful for programmers who use the **make** feature. If you are not a programmer you will probably not use this command much. You can also create files using editors such as **emacs**, **vi**, **pico**, etc. There are a lot of editors available. To create a directory you can use the **mkdir** command:

```
bash$ mkdir temp
```

This created a directory called **temp** that you can now copy files into.

4.4.2 Moving Files

Moving files is easier then it might sound. In order to move a file called **newfile.txt** into a directory called **temp/** you could type:

```
bash$ mv newfile.txt temp/
```

If the directory did not exist then this would change the name of the file from **newfile.txt** to **temp/** instead. If you are logged in as root or have superuser privileges you can copy the file to a directory that the user might not have rights to copy files to:

```
bash# mv newfile2.txt /var/www/html/
```

4.4.3 Deleting Files

In order to delete a file you can type **rm** before the file name:

```
bash$ rm newfile3.txt
```

You can also delete a whole directory and all of the files in it including sub-directories with the **rm** command with the **-R** switch:

```
bash$ rm temp -R
```

4.4.4 Copying Files

You can copy files using the **cp** command. This command can either duplicate the file in the same directory or copy the file to another directory. First how to duplicate a file in the same directory:

```
bash$ cp newfile5.txt newfile6.txt
```

You can now create a directory and copy the file into it:

```
bash$ mkdir temp2
bash$ cp newfile5.txt temp2
```

4.5 Viewing and Editing Files

4.5.1 Text Viewers

To view the contents of a file you can use the commands **cat**, **head**, **tail**, **more**, and **less**. These commands are pretty easy to use, so we will do a few examples. There are two dictionary files in the **/usr/share/dict** directory called **linux.words** and **words**. We are going to take a look at these two files using the viewing commands. First **cat**:

```
bash$ cat /usr/share/dict/linux.words
```

I would put the results, but that would require quite a few lines. Well, **head** is next:

```
bash$ head /usr/share/dict/linux.words
Aarhus
Aaron
Ababa
aback
abaft
abandon
abandoned
abandoning
abandonment
abandons
```

This one was not as long as the last one. The **head** command prints the first 10 lines of the file to the screen by default. If you only wanted the first three lines you could type this:

```
bash$ head -n 3 /usr/share/dict/linux.words
Aarhus
Aaron
Ababa
```

You can also do the same with **tail** for the end of the file:

```
bash$ tail /usr/share/dict/linux.words
zoologically
zoom
zooms
zoos
Zorn
Zoroaster
Zoroastrian
Zulu
Zulus
Zurich
```

Like **head** with **tail** you can say the number of lines you would like to display:

```
bash$ tail -n 3 /usr/share/dict/linux.words
Zulu
Zulus
Zurich
```

This is good when you would like to look at a log file or something else that is long, but you only want to see the last few lines of the file. The commands **more** and **less** are very similar. You start **more** the same way as **cat**, **head**, and **tail**:

```
bash$ more /usr/share/dict/linux.words
```

This will display one screen of text at a time. With **more** you can press the **spacebar** to scroll down a screen at a time and the **Enter** key to go down a line at a time. The biggest problem with **more** is that you cannot scroll back up. This is where **less** is very strong:

```
bash$ less /usr/share/dict/linux.words
```

This will also display a screen of text at a time. The up and down arrows go up and down a line at a time. You can also press **Page Up** and **Page Down** to scroll a page at a time. With both **more** and **less** you can press **q** to quit early. Output of programs can be piped into these programs also. This is how you would pipe the output of the **ls** command into **less**:

```
bash$ ls /dev/ | less
```

This would of course put a lot of text on the screen so we will not put it here. With these commands you can see many text files better then you would have been able to before.

4.5.2 Editors

There are three text editors that I think you should know about. **emacs**, **pico**, and **vi** are all on the Red Hat Linux distribution and they are also on other versions of UNIX and even other well known operating systems. Editors are sometime hard to learn, but can be very powerful and even necessary for system administrators.

To use **emacs** you can type **emacs** followed by the filename. To create or modify a file called **text1.txt** you would type the following:

```
bash$ emacs text1.txt
```

This will take you into the **emacs** editor. Here are a few commands that you might find very useful:

```
Ctrl-x Ctrl-s        Save the file
Ctrl-x Ctrl-c        Exit Emacs
Ctrl-k               Cut a line
Ctrl-y               Paste cut lines
Ctrl-h t             Emacs Tutorial
```

If you want to learn **emacs** take the tutorial and have fun.

The University of Washington created a mail reader called **pine**. With it they also created a text editor called **pico**. Pico is easy for starters to get used to, but does not seem to have very much power as a text editor. To create a file called **text2.txt** type:

```
bash$ pico text2.txt
```

This will take you into the editor. You can there type whatever you want to type, then you can use the commands on the bottom of the screen to do whatever you want. Here are a few commands:

```
Ctrl-o              Save the file
Ctrl-x              Exit pico
Ctrl-k              Cut a line
Ctrl-u              Paste cut lines
Ctrl-t              Spell Checker
```

Pico is a pretty easy to use, small editor that does not require as many libraries loaded to run as **emacs**. This usually does not matter, but if you ever find yourself in linux rescue mode, **pico** will work while **emacs** will not.

The **vi** editor is one of the oldest and most available editors around. **emacs** is probably more used today, but on those old machines that had the Y2K problem **vi** is probably available where **emacs** may or may not. You can create a file with **vi** by typing:

```
bash$ vi text3.txt
```

This will take you into the **vi** editor, but you must type in commands before you can insert any text. **vi** has two or three different modes. You start in command mode. From here you can enter text mode or do things like save or edit your document. Here are a few command mode options:

```
:w                  Save the file
:q                  Exit vi
x                   Delete the current character
i                   Enter Insert Mode
a                   Enter Insert Mode after the current character
```

When you are in insert mode you can exit by pressing the Escape (**ESC**) key.

4.6 File Permissions

File permissions are an important part of any UNIX environment. You have read, write, and execute permissions for each set of users, the owner, the files group, and everyone else on the system. If you take a long listing of a directory you can see what these permissions are set to:

```
bash$ ls -al
total 48
drwx------    4 bob      bob          4096 Feb 26 10:33 .
drwxr-xr-x    5 root     root         4096 Feb 26 10:33 ..
-rw-r--r--    1 bob      bob            24 Feb 26 10:33 .bash_logout
-rw-r--r--    1 bob      bob           224 Feb 26 10:33 .bash_profile
-rw-r--r--    1 bob      bob           124 Feb 26 10:33 .bashrc
-rw-r--r--    1 bob      bob          5450 Feb 26 10:33 .canna
drwxr-xr-x    2 bob      bob          4096 Feb 26 10:33 Desktop
-rw-r--r--    1 bob      bob           747 Feb 26 10:33 .emacs
drwxr-xr-x    3 bob      bob          4096 Feb 26 10:33 .kde
-rw-r--r--    1 bob      bob          3728 Feb 26 10:33 .screenrc
-r--r--r--    1 bob      bob          1019 Feb 26 10:33 .wl
```

In this directory the file **.emacs** has a string of 10 characters at the beginning of the line. The first character is where you would have information about what

type of file this is. The **d** character stands for directory. The **l** character stands for symbolic link. After the first character there are nine characters left. These nine characters are split into three groups of three. The first three characters are the permissions for the owner. The next three are the permissions for the files group, and the last set of permissions are the permissions for everyone else on the system. (Everyone else on the system includes the users like apache the web server.)

Each set of three characters are split into three permissions read, write, and execute. These are marked with a letter if the permission is available or marked with a dash if the permission is not set.

4.6.1 Read Permissions

Read permissions allow the user to view the contents of the file. If you have read permissions, but you do not have execute permissions you can copy the file into another directory and change the permissions of the file. You can also change the permissions of the file if you have permissions for that directory. You can give everyone read access to the file by typing:

```
bash$ chmod a+r text1.txt
```

You can give permissions to only the user, group, or others by using **u**, **g**, and **o** instead of **a**. This is what you would do to give read permissions to the group:

```
bash$ chmod g+r text1.txt
```

If you wanted to take away everyones read permissions you would replace the plus sign with a minus sign:

```
bash$ chmod a-r text1.txt
```

You can also set the file by typing the binary values of each bit. We will talk about that later in this section.

4.6.2 Write Permissions

Write permissions allow the user to write information to the file. The user may or may not have permissions set to view the contents of this file. If you are creating a log that you do not intend others to read, but you would like them to be able to add to you could create a write only file. If you wanted people to be able to read the file, but not modify it you could take away the write permissions and make it a read only file. Note that if a user has permissions to write to a file, then the user can erase the file also. To give write access to everyone type:

```
bash$ chmod a+w text2.txt
```

If you wanted to take away everyones write permissions you would replace the plus sign with a minus sign:

```
bash$ chmod a-w text2.txt
```

4.6.3 Execute Permissions

Execute permissions give the user the right to run the code in the file. This does not always allow the code to be run. If the code is a shell script then the user needs to have read permissions in order to run the program. This gets a little bit complex. Usually execute permissions go with read permissions for executable programs. Directories also have an execute bit set if you are allowed to enter the directory. You may enter, but not see anything there unless the read bit is set. To give execute permissions to a file type:

```
bash$ chmod a+x text3.txt
```

If you wanted to take away everyones execute right you would replace the plus sign with a minus sign:

```
bash$ chmod a-x text3.txt
```

I hope that you are beginning to see a pattern. **chmod** is not really that hard to use.

4.6.4 Expressing Permissions in Binary

The three sets of three characters are really 9 binary bits that mark 1 for yes you have the right and 0 for no, you do not. These bits are displayed in octal. If you do not know what that means then just assume that they are represented by three digits from 0 to 7, each digit representing a different set of three bits. If you wanted to give read, write, and execute rights you could represent that in binary as 111 or in octal as 7. To convert from octal assume that the first bit stands for 4, the second for 2, and the last for 1. If you can do this then you will be able to figure out what permissions to use. For a file that is read only for everyone you would use 444. For a file that was write only for everyone you would use 222. If the file was execute only for everyone the permissions would be set to 111. If you want the owner to have read, write, and execute and everyone else to have read and execute rights you would use 755. Web pages would usually be set to 644 so that the owner can read and write, but everyone else can only read them:

```
bash$ chmod 644 index.html
```

For CGI files you would have to use 755 so that the world can read and execute them:

```
bash$ chmod 755 results.cgi
```

I mentioned that there are only 9 binary bits to set permissions with, but that is not exactly correct. There are other bits that you can use to do things like setuid and setgid. With setuid the executable runs as the owner. With setgid the executable runs as the group owner. This can be accomplished by putting either a 4 (for setuid) or a 2 (for setgid) in front of the other permissions. If you wanted **useradd** to run as root, allowing anyone to create users you could change permissions like this:

```
bash# chmod 4755 /usr/sbin/useradd
```

If you then looked at the directory in list format, the line with **useradd** would look like this:

```
-rwsr-xr-x    1 root  root   52348 Mar  9  2001 /usr/sbin/useradd
```

Notice the **s** instead of the **x** in the first three letters. The **s** stands for setuid.

4.7 Finding Files

If you are looking for a file there are a couple of ways of doing it. The easiest way is to use the **locate** command:

```
bash$ locate useradd
/usr/share/man/man8/useradd.8.gz
/usr/share/man/pl/man8/useradd.8.gz
/usr/sbin/useradd
```

This sometimes gives you more information than you really wanted. If you were looking for the **useradd** command to add users, then you found it, but you also found a couple of manual pages. If there were a lot of files with **useradd** in the names then you would have a long list. The next command you could try is **which**:

```
bash$ which useradd
/usr/sbin/useradd
```

This helps you to know where the file is, but it only works for executable files in your path. If you were looking for a text file then you would be better off with **locate.** What if you wanted to look for a file that was owned by someone, but you did not know what the name was? You could use the command **find**:

```
bash$ find /home/ -user bob
find: /home/dave: Permission denied
/home/bob
/home/bob/.bash_logout
/home/bob/.bash_profile
/home/bob/.bashrc
/home/bob/.canna
/home/bob/.kde
/home/bob/.kde/Autostart
/home/bob/.kde/Autostart/Autorun.desktop
/home/bob/.kde/Autostart/.directory
/home/bob/Desktop
/home/bob/Desktop/kontrol-panel
/home/bob/Desktop/.directory
/home/bob/Desktop/Linux Documentation
/home/bob/Desktop/www.redhat.com
/home/bob/Desktop/Printer
/home/bob/.emacs
```

```
/home/bob/.screenrc
/home/bob/.wl
/home/bob/.xauth
/home/bob/.bash_history
```

This found all of the files that Bob owned that were in **/home** or any of its subdirectories. You will probably notice that the first line was not a file. This line is an error messages. You can get rid of the error messages by using a redirection of standard error to the **/dev/null** device:

```
bash$ find /home/ -user bob 2> /dev/null
/home/bob
/home/bob/.bash_logout
/home/bob/.bash_profile
/home/bob/.bashrc
/home/bob/.canna
/home/bob/.kde
/home/bob/.kde/Autostart
/home/bob/.kde/Autostart/Autorun.desktop
/home/bob/.kde/Autostart/.directory
/home/bob/Desktop
/home/bob/Desktop/kontrol-panel
/home/bob/Desktop/.directory
/home/bob/Desktop/Linux Documentation
/home/bob/Desktop/www.redhat.com
/home/bob/Desktop/Printer
/home/bob/.emacs
/home/bob/.screenrc
/home/bob/.wl
/home/bob/.xauth
/home/bob/.bash_history
```

This is a very powerful command, but it takes a little getting used to. If you know what directory a file is in and you know what you are looking for you might want to check out the powers of **grep**. Grep searches the contents of each file and matches strings. Here is a a line that you could use to search for the word "bob" in files in the **/etc/** directory. Notice that we will again redirect standard error to the **/dev/null** device:

```
bash$ grep bob /etc/* 2> /dev/null
/etc/group:bob:x:502:
Binary file /etc/ld.so.cache matches
/etc/passwd:bob:x:502:502::/home/bob:/bin/bash
/etc/termcap:bobcat|sbobcat|HP 9000 model 300 console:\
/etc/termcap:# * Added 'bobcat' and 'gator' HP consoles and the Nu machine entries
/etc/termcap:#   hpgeneric, hpansi, hpsub, hp236, hp700-wy, bobcat, dku7003, adm11,
```

Well, that is a lot of information. It lists a few files and the line that has the three letter string "bob". You can search the immediate subdirectories of **/etc/** in almost the same way:

```
bash$ grep bob /etc/*/* 2> /dev/null
```

Unfortunately, there is nothing there. Hopefully these four commands will help in finding files.

4.8 Information About the System

Sometimes it is important to know about your system and the environment in which you are working. Seeing the environment is quite easy. You can just type:

```
bash$ env
```

This of course is a little bit long and hard to read. For specific information there are often simple commands. To know which user you are logged in as you can type:

```
bash$ whoami
bob
```

That is nice, but what machine am I logged into?

```
bash$ hostname
bob.bobnet.com
```

Well, what about my processor and other things like that? You can see a lot of information in your **/proc/** directory. **/proc/** is not a normal directory and is sometimes very strange. Files are constantly updated there. This is really, in a way, a shell interface with the kernel. Let's look at some statistics listed inside of the **/proc/cpuinfo** file:

```
bash$ cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 8
model name : Pentium III (Coppermine)
stepping : 1
cpu MHz : 598.632
cache size : 256 KB
fdiv_bug : no
hlt_bug : no
f00f_bug : no
coma_bug : no
fpu : yes
fpu_exception : yes
cpuid level : 2
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov
pat pse36 mmx fxsr sse
bogomips : 1196.03
```

This tells you a lot of information about the first processor named processor 0. If you had a second processor it would be marked as processor 1 and would have all of the same information. You can also look at your memory information by typing:

```
bash$ cat /proc/meminfo
```

Information in **/proc/meminfo** is about as long as the **cpuinfo** file and it lists information that may seem a little bit more complex. With this information you can know a lot about your hardware and how it is being used.

You can also know about the file partitions and the space that they are using with the **df** command:

```
bash$ df
Filesystem       1k-blocks      Used Available Use% Mounted on
/dev/hda5         38978244   5017232  31981004  14% /
/dev/hda1            46636      3858     40370   9% /boot
/dev/hda2          5036316     87012   4693472   2% /var
/dev/fd0              1423      1405        18  99% /mnt/floppy
```

This information is useful when you are wondering how much of the system you have used and how much harddisk is left.

4.9 Basic Shell Scripts

Basic shell scripts are files that contain a list of commands that the shell executes one after another. More advanced shell scripts do tests and run commands differently depending on the results of the tests. In this section you will see the basics of creating a script to do something.

4.9.1 Create the script

Using one of the editors covered in Section 4.5.2 create a file called **datedir.sh** that contains the following lines:

```
date > logfile
ls -al >> logfile
```

This is going to become a script that will write the date and the contents of the current directory into a file called **logfile**.

4.9.2 Change Permissions and Execute

Once the file is created you can change the permissions of the file so that only you can execute the file. You can do this using chmod:

```
bash$ chmod 700 datedir.sh
```

Now that you have set the permissions so that the shell script can execute, let's run it:

```
bash$ ./datedir.sh
```

Since this file wrote all of its contents to another file called **logfile**, we can see the contents of the file by using **cat** to view it:

```
bash$ cat logfile
Wed Mar 20 17:39:54 HST 2002
total 64
drwx------    5 bob      bob      4096 Mar 20 17:39 .
drwxr-xr-x    5 root     root     4096 Feb 26 10:33 ..
-rw-------    1 bob      bob       416 Mar 19 16:34 .bash_history
-rw-r--r--    1 bob      bob        24 Feb 26 10:33 .bash_logout
-rw-r--r--    1 bob      bob       224 Feb 26 10:33 .bash_profile
-rw-r--r--    1 bob      bob       124 Feb 26 10:33 .bashrc
-rw-r--r--    1 bob      bob      5450 Feb 26 10:33 .canna
-rwx------    1 bob      bob        33 Mar 20 17:39 datedir.sh
drwxr-xr-x    2 bob      bob      4096 Feb 26 10:33 Desktop
-rw-r--r--    1 bob      bob       747 Feb 26 10:33 .emacs
drwxr-xr-x    3 bob      bob      4096 Feb 26 10:33 .kde
-rw-r--r--    1 bob      bob        29 Mar 20 17:39 logfile
-rw-r--r--    1 bob      bob      3728 Feb 26 10:33 .screenrc
-r--r--r--    1 bob      bob      1019 Feb 26 10:33 .wl
drwx------    2 bob      root     4096 Mar 19 15:38 .xauth
```

You can see that this file contains the date as the first line, then the directory listing is appended onto the end. You can change this shell script so that it does a backup or many other things.

Chapter 5

Setting up Mail

In this chapter we will assume that the mail server is already running and you just want to get mail working on your own machine. If you do not have a mail server running and you want to do that first, skip ahead to Chapter 13.

5.1 Mail Sources

There are really two main sources of mail. Either your machine gets mail directly or the mail is stored on a mail server. If the mail is stored on your machine then getting the mail will be really easy. If it is on a different machine then you will need to know what type of mail protocol you will have to use to get mail from the mail server.

5.1.1 Local Mail

The default directory to store mail is **/var/spool/mail/** on many Linux machines. Another place that you can often find your mail is in your home directory in a file called **inbox** or **mbox**. Most mail readers can read mail on your machine without a problem.

5.1.2 Mail Servers

Mail servers receive mail for you and hold it until you connect and get your mail through a protocol like POP or IMAP. Sometimes you can have the mail servers forward the mail to your machine or deliver it to a directory that you have mounted on your machine. Sometimes it is better to have the mail delivered to your machine than let it sit on the mail server until you go and get it. If you have a directory on the mail server then you can probably setup a **.forward** or **.procmailrc** file to forward the mail to you or put it in a file for you.

5.2 .forward and .procmailrc

The **.forward** and **.procmailrc** files help you get mail to where you want it to be. The **.forward** file is the one that you will find available on more systems, but the **.procmailrc** provides you with more options. It is probably best to

learn both and use the one that you think is the best for your purposes. Once you have decided to create one of these files you will have to put the file in your home directory.

5.2.1 .forward files

Using the **.forward** file to handle mail you can do two basic things. You can forward the email message to another account or person or you can send the email message as the standard input for a program. In a **.forward** file you can do more than just one of these. You can send email to a list of people and send the email as input to a program. Here is what the **.forward** file would look like to forward the email to another account:

```
mike@bob.org
```

Okay, that was simple. Maybe you want both Bob and Mike to receive the email message:

```
mike@bob.org
bob@bob.org
```

To get the email message directed into the input of a program create a **.forward** file that looks like this:

```
|/home/bob/mailfilter.pl
```

You would need to have a program to handle the mail, but if you have decided that you want to run the email through a program, you probably already have a program to use, or you probably know how to make one.

5.2.2 .procmailrc files

The **.procmailrc** file can do everything that the **.forward** file can do, but the **.procmailrc** file has a few added options. Here is the simple email forwarding:

```
:0
! mike@bob.org
```

Extra recipients can be added easily:

```
:0
! mike@bob.org

:0
! bob@bob.org
```

When you write your **.procmailrc** or **.forward** file you should make sure that you do not loop the message. If the message gets sent back to your self that could cause problems. Now for sending mail to a file:

```
:0
mbox
```

Sending a message to a program is the basically the same as in the **.forward** file:

```
:0
|/home/bob/mailfilter.pl
```

Now that you have the basics, you can add options to sort your mail. To forward mail with mike in the subject line to Mike use these lines:

```
:0
* ^Subject:.*mike
! mike@bob.org
```

If you are having problems with getting mail forwarded to you that you do not want, you can trash it like this:

```
:0
* ^To:.*specialoffer@maileveryone.com
/dev/null
```

If you would like to learn more about **.procmailrc** and the options that you have, look at the man page:

```
bash$ man procmail
```

5.3 mail

The **mail** command is one of the oldest commands. If you are on a machine that receives mail you will most likely have the **mail** command. It is probably not the most easy command to use, but it is a good one to know the basics of. To read your mail type:

```
bash$ mail
```

If your mail is stored in a file and the **mail** command does not open it automatically then you might need to specify the file to open:

```
bash$ mail -f /home/bob/mbox
```

Once you have **mail** open you will see a list of subjects and numbers. You can type the number of the message to view it. Once you have read the message you can press **q** to quit, **n** for your next message, **d** to delete, or **?** to see your available options. When you exit mail will usually store the messages you read in your home directory in a file called **mbox**. To send mail type:

```
bash$ mail mike@bob.org
Subject:
```

At the subject prompt type the subject in and press **Enter**. Then type your message until you have finished. Once you are done you can press either **Ctrl-d** or **.** on a line by itself to finish. Then you can add in the other people you want to carbon-copy the email message to.

5.4 emacs and rmail

To use **emacs** and **rmail** you first need to start **emacs**:

```
bash$ emacs
```

After emacs is running press a key to get to the scratch window. From there press **ESC**. Wait until **emacs** says ESC- in the lower left hand corner. Then press **x** and type **rmail**:

```
M-x rmail
```

You can use the keys **p**, **n**, **g**, **d**, **m** to go to the previous message, next message, get new messages, delete messages, and create a message (in half of the window) respectively. After you have created a message you can press **Ctrl-c Ctrl-c** to send the message. If you still have the window split into two then you can press **Ctrl-x o** to switch in between the two windows. If you press **Ctrl-x 1** then the window that is currently active will take over and the others will disappear. To quit **rmail** just press **q**. Then exit **emacs** normally (**Ctrl-x Ctrl-c**).

5.5 Netscape Mail

Using Netscape mail is not very difficult. You need to know where you are getting your mail from before you can get it though. In Netscape select Preferences from the Edit menu. You need to setup Identity and Mail Servers.

5.5.1 Identity

You need to put in a name and an email address in for identity. Once those two are in you will be able to send mail using Netscape. The email address cannot just be your login name. You must have a full and complete email address.

5.5.2 Mail Servers

If you are getting mail locally on your machine then you will need to use the **MoveMail** Server type. All you need to do is set your User Name to be your login name. This will move your mail from the default location at something like **/var/spool/mail/bob** to a place where Netscape can read it easier.

If you need to use either a **POP** server or **IMAP** server then you will need to select the server type and then enter in the hostname or IP address of the mail server. If you are not sure where the mail comes to then you can ask someone else who has set up mail on the same machine.

Once you have your incoming mail settings figured out you will need to setup the outgoing settings. If your mail server is on the same machine you can just type in **localhost** in for the mail server. If the mail server is is separate machine then you will need to type in the hostname or IP address. For the outgoing mail server user name you will again need to put in your login name.

5.6 Pine

pine is an easy to use text menu based mail client. If you are receiving mail locally on your machine you can just type **pine** to send and receive mail. If you are not then it might be a lot more difficult. **pine** does not work well with **POP** servers, but can connect to **IMAP** servers. In **pine** you have a list of commands at the bottom of the screen. Here is a quick list of commands from the main menu:

```
?                   Help
c                   Compose a message
i                   Message Index
l                   Folders
a                   Address Book
s                   Setup
q                   Quit
```

When you are composing an email message the commands are almost the same as the **pico** editor commands. When you are finished with the message you can press **Ctrl-x** to send it.

Chapter 6

Software Installation

It is quite likely that you will have a time when you want to install software. Once you have decided that you would like to install some software, you need to know what kind of permissions you have on the system. If you are a root user you can install the easy **.rpm** packages, but if you do not have root privledges you have to settle for other formats.

6.1 Choosing File Formats

Not all software available on the Internet can run on your machine. Most people installing their own systems have Intel processors. When you are looking at files to download it is quite likely that you will see files like the following:

```
package.tar.gz
package.tgz
package.bz2
package.tbz2
package.zip
package.rpm
package.i386.rpm
package.SPARC.rpm
package.Z
package.tar
package.c
package.bin
```

Okay, there are a lot of file types. Usually files that end with **.tar**, **.tar.gz**, **.tgz**, **.Z**, and **.bz2** are source code. This is not always the case, but often it is. Files with the **.zip** extension are usually compiled Windows files and usually are not easy to use. The **.rpm** extension stands for Red Hat Package Manager and these require root to install. The **.c** extension usually means that the file is C source code. If you are on a machine with an Intel processor you probably want to look for packages with **i386**, **i586**, or **i686** in the name. These packages probably have compiled Intel code. Words like **SPARC** or **Alpha** usually mean that the package is already compiled on that specific processor chip and will not run on the Intel processor. Here is the list of files again with a command to start the extraction:

```
package.tar.gz      tar xvfz package.tar.gz (Might need to compile)
package.tgz         tar xvfz package.tgz (Might need to compile)
package.bz2         bunzip2 package.bz2 (Might need to compile)
package.tbz2        bunzip2 package.tbz2; tar xvf package.tar (Might need to compile)
package.zip         unzip package.zip
package.rpm         rpm -i package.rpm
package.i386.rpm    rpm -i package.i386.rpm
package.SPARC.rpm   (Not for Intel Processor)
package.Z           uncompress package.Z (Might need to compile)
package.tar         tar xvf package.tar (Might need to compile)
package.c           gcc package.c -o package
package.bin         sh package.bin (or ./package.bin)
```

6.2 Compiling Source Code

Compiling is always difficult unless you know about programming. Not all software is easily portable, but usually the programmer writes information in the code about how each system should compile the code. The best way to compile the code is to look for and read a file called either **README** or **INSTALL**. Also look for a file called **Makefile**. Once in the directory with the files try reading the files. If they do not help try typing these lines:

```
bash$ ./configure
bash$ make
bash$ make all
bash$ make install
```

If one of those commands does something then that might be a good sign. If they do not then you will probably have to look around the package for some documentation. The only way to learn about installing software is to read.

Chapter 7

Administration Tools

Administrators need to know how to change things and how they work. For every Operating System there are tools to make the administration job easier. In Linux there are many good tools to use. The list of available tools changes with every release of Red Hat. The important ones that look permanent are discussed below.

7.1 Setup Command

The **setup** command has the generic name that you would expect a powerful tool to have. This command is run from the shell while you are logged in as root. Once you type this command you will get a menu of the available options. This is similar to what the list looks like:

```
lqqqqqqqqu Choose a Tool tqqqqqqqqqk
x                                  x
x   Authentication configuration   x
x   Firewall configuration         x
x   Keyboard configuration         x
x   Mouse configuration            x
x   Network configuration          x
x   System services                x
x   Sound card configuration       x
x   Timezone configuration         x
x   X configuration                x
x                                  x
x      lqqqqqqqqqqk  lqqqqqqk      x
x      x Run Tool x  x Quit x      x
x      mqqqqqqqqqqj  mqqqqqqj      x
x                                  x
x                                  x
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
```

These are basically all self expanitory, so you should feel free to experiment and test them out. Most of them are what you see when you install Linux in text mode.

7.1.1 Authentication Configuration

If you select Authentication Configuration you will be taken to a menu where you will be able to select a password server. In order to setup NIS, LDAP, or Hesiod you will need to have at least one of those servers running. Later in this book we will talk about setting up an NIS server. If you would like to learn more skip ahead to Chapter 14.

After selecting **Next** you will be taken to a menu where you can decide if you want to use Shadow passwords, MD5 passwords, LDAP Authentication, or Kerberos 5.

Shadow passwords only means that the passwords will be stored in the **/etc/shadow** file instead of the **/etc/passwd** file. This is good because password crackers will have a slightly more difficult time getting the passwords to run a cracking program against.

MD5 passwords are longer. Usually you can only have 8 characters in a password, but with MD5 the passwords can be up to 256 chatacters long. With out MD5 these passwords would all be the same:

```
abcdefgh1
abcdefghijklmn
abcdefgh987654321
abcdefghABCDEFGH
```

LDAP and Kerberos are both server related options. LDAP servers provide passwords and directory space. Kerberos servers give tickets that allow you to authenticate once to a server, then continue logging into other Kerberos machines without reauthenticating.

The authentication menu can also be started by running **authconfig** from the shell.

7.1.2 Firewall Configuration

This firewall configuration tool uses **ipchains** to setup the firewall. You can also start this tool by typing **lokkit** in the shell. Once it is started you are given four choices. You can choose from High, Medium, No Firewall, and Custom.

High firewall blocks about everything. If you are using High you are severely limited in what you can do. You will not be able to run servers if you have the firewall set to high because the firewall will block all connections to the ports that run servers.

Medium is nice, but like High, you will not be able to run servers. I recommend not choosing either of these two.

No Firewall is the easiest for someone to run, but that leaves everything open. Being open is not always a bad thing if you are running a server. If the server is set to No Firewall then you will probably want to configure the firewall later using a tool like **firewall-config**.

If you do a custom firewall then you will be taken to another menu. On this menu there will be two sections. Trusted devices are ones that will not be blocked at all. Allow incoming has some common services listed by name and a space to put additional ports. Move the curser with the tab key and select

with the space key. If you want to open a service, but do not know which port it runs on you can try looking in the **/etc/services** file and see if the service is listed.

When this utility is done it will store an **ipchains** configuration file at the location **/etc/sysconfig/ipchains**. You can go there and modify the file later if you like.

7.1.3 Keyboard Configuration

Keyboard Configuration just allows you to pick the type of keyboard that you use. You just look at the list and pick the country that your keyboard was made for. That is about it. You can also start this configuration by typing **kbdconfig** in the shell.

7.1.4 Mouse Configuration

Mouse Configuration can also be started by typing **mouseconfig**. Once in the menu you can choose the mouse that you are using. It is usually best to choose the generic mouse so that you can change it anytime it goes bad without having to worry about changing the mouse settings. I have noticed that sometimes if the mouse setting is incorrect then the X windows display crashes.

7.1.5 Network Configuration

Network configuration is really good if you are only using one network card and you do not want to do much with your machine as a router, etc. Here you can decide between static network settings and the dynamic DHCP settings. DHCP is good for workstations that get their IP address and other information from a DHCP server on the same network.

7.1.6 System Services

System Services are the services that start when your machine boots up. You can also setup these services with the **chkconfig** command that is covered later in this chapter. You can go down the list and select or unselect services with the **spacebar.** If you are wondering what a service is or what it does you can select the service and press **F1** for a little bit of information. You can start this service from the shell by typing **ntsysv**.

7.1.7 Soundcard Configuration

The soundcard configuration will try to probe your sound card. Once it thinks it knows what your soundcard is it will play sound to test the soundcard. If the probe is not successful then you can manually configure the sound card by running the tool from the shell with the noprobe option:

```
bash# sndconfig --noprobe
```

7.1.8 Timezone Configuration

Timezone Configuration allows you to choose the timezone that you are in. This is usually not that important to some people, but if you decide to get time from a network time server then the time will be converted into your timezone (the one you select here). This utility can also be run from the shell by typing **timeconfig**.

7.1.9 X Configuration

In X configuration you select the video card and monitor that you are using. Sometimes your selections do not seem to work. There are a lot of video drivers for Linux, but there is still a lot of development going on right now. If your card is not supported it is likely that something else will work. Usually the probing does it correctly, but if not you can select and test until something works.

7.2 System Services

System Services are the things that run in the background and keep the system doing stuff even when nothing on the screen is moving. Services such as web servers, timed events, and firewalls all run in the background.

7.2.1 Symbolic Links

You can create a symbolic link to a service so that it starts or stops at different run levels. Symbolic links do all of the database work required to know which servers to start and which ones to stop.

If you go into the **/etc/rc.d/** directory you will find a lot of subdirectories and some files. It should look something like this:

```
bash# ls -l
total 60
drwxr-xr-x    2 root     root         4096 Jan 22 16:54 init.d
-rwxr-xr-x    1 root     root         3047 Feb  7  2001 rc
drwxr-xr-x    2 root     root         4096 Oct 29  2001 rc0.d
drwxr-xr-x    2 root     root         4096 Oct 29  2001 rc1.d
drwxr-xr-x    2 root     root         4096 Oct 29  2001 rc2.d
drwxr-xr-x    2 root     root         4096 May 21 11:31 rc3.d
drwxr-xr-x    2 root     root         4096 May 21 11:31 rc4.d
drwxr-xr-x    2 root     root         4096 May 21 11:31 rc5.d
drwxr-xr-x    2 root     root         4096 Oct 29  2001 rc6.d
-rwxr-xr-x    1 root     root          962 Jan 29  2001 rc.local
-rwxr-xr-x    1 root     root        18851 Apr  7  2001 rc.sysinit
```

The files **rc**, **rc.local**, and **rc.sysinit** are programs that help with runlevel changes.

rc starts and stops services as the runlevel changes. **rc.local** runs after all of the rest of the initialization scripts. If you want something started after everything else, you can put it on the end of this file. **rc.sysinit** runs once at boot time and starts getting everything set up the way it should be.

OK, now for the symbolic links. If you go into one of the directories with a digit in them such as **rc5.d/** you will be able to see what happens when this runlevel is started.

The files in each of these directory are links that point to the real files in the **/etc/rc.d/init.d/** directory. The links either start with **K** or **S**. **K** stands for Kill and **S** for Start. After the letter comes a two digit number. The letters and numbers are sorted when the runlevel is started and each service that has a **K** in front is killed if it is running and everything with an **S** in front is started if it is not already running.

If you wanted to change the order in which something is started or stopped you could change the two digit number to something else. Lower numbers come before higher numbers because they are alphabetically sorted.

To create a link you can use the **ln** command. If you were in the **/etc/rc.d/rc5.d/** directory and wanted the **httpd** server to start at about 85 you would type this:

```
bash# ln -s ../init.d/httpd S85httpd
```

This will create a link called **S85httpd** that points to the real file found at **/etc/rc.d/init.d/httpd**.

7.2.2 Using chkconfig

Creating all of the symbolic links can be a lot of work, especially when you realize that for each change in a service you might have to change up to 7 symbolic links. You have to create either a Kill or a Start link for every service at every runlevel.

The tool **chkconfig** does this for you. You can list all of the services and what they do at each runlevel by typing:

```
bash# chkconfig --list
```

This will list a lot of services so you might have to scroll through the list. If the **xinetd** service is on then you will get a list of all of the subservices that it provides. All of these services will run at the same runlevel as **xinetd**.

If I wanted a the mail server to run on startup I would type this:

```
bash# chkconfig sendmail on
```

This will start **sendmail** at the runlevels and the order listed at the top of the **/etc/rc.d/init.d/sendmail** file. The line looks something like this:

```
# chkconfig: 2345 80 30
```

This means it will run at levels 2, 3, 4, and 5. It will start as number 80 and stop as number 30.

For more information about **chkconfig** try either typing the name or looking at the manual page:

```
bash# chkconfig
bash# man chkconfig
```

7.3 User Accounts

User Accounts are very important to machines that have more than one user. If you decide all of the users names you can set everything up when you install Linux, but if you have not decided everyone by then, you will have to manage accounts by hand.

7.3.1 Account Creation/Deletion

There are easy commands for creation and deletion of user accounts. To create an account you can use the **useradd** command. Here is the easiest example:

```
bash# useradd bob
```

This will create a user account called bob. If you want to insert more information you can. Take a look at this example:

```
bash# useradd -c ''John Doe'' john
```

This adds a little bit of information. This comment is stored in the **/etc/passwd** file.

To delete an account you can type:

```
bash# userdel bob
```

This will erase the account, but the directories will still be there. To erase everything in the home directory use this command instead:

```
bash# userdel -r john
```

7.3.2 Setting/Forgotten Passwords

The accounts above do not yet have passwords. To set a user password for bob use this command:

```
bash# passwd bob
```

You will be asked to type the password twice. As long as the passwords match the password will be changed. The superuser (root) can set the password to things that the users cannot.

Chapter 8

Data Backup

8.1 File Archives and Compression

Making a compressed file archive is good for two reasons. The more obvious reason is to save space. The second and less obvious reason is because with a compressed archive file you are able to group many files into a single file that is easier to work with. Assuming you put a lot of files in a directory and you want to get all of the files in the directory into a single file, here is a table of the commands you could use, and the commands to expand or uncompress:

Grouping/Compression Command	Resulting File	Expanding Command
tar cvf file.tar directory/	file.tar	tar xvf file.tar
tar cvfz file.tar.gz directory/	file.tar.gz	tar xvf file.tar.gz
zip file.zip directory/	file.zip	unzip file.zip
bzip2 file.tar	file.tar.bz2	bunzip2 file.tar.bz2
compress file	file.Z	uncompress file.Z

This table should help you with compression and file archives.

8.2 Using Floppies

Floppy diskettes are great for many things. Their main problem is that they are small. Linux machines use a different default file system from Microsoft Windows, so as you might guess, their floppies have a different formatting. Although the diskettes can have different formatting, Linux is able to read and format disks into Microsoft format.

8.2.1 Formatting Floppies

In order to format a floppy there are two main commands I would suggest. If you are using a GUI you should look into using **gfloppy**. This command gives you a graphical interface where you can select the file system type for the disk and format it. If you are using the shell then you can use **mke2fs**. This is what you would type to format a floppy diskette:

```
bash# mke2fs /dev/fd0
```

In this command the **/dev/fd0** refers to the floppy device numbered zero (the first one.) If you have more than one floppy you could change the floppy device number and format in another drive:

```
bash# mke2fs /dev/fd1
```

This will format the diskette to the ext2 file system format.

8.2.2 Reading and Writing Floppies

To read from and write to floppies is pretty easy, but you have to get used to it. There are a whole suite of DOS related tools called **mtools**. You can see a complete list of the available tools by tools by typing:

```
bash$ mtools
Supported commands:
mattrib, mbadblocks, mcat, mcd, mcopy, mdel, mdeltree, mdir
mdoctorfat, mdu, mformat, minfo, mlabel, mmd, mmount, mpartition
mrd, mread, mmove, mren, mshowfat, mtoolstest, mtype, mwrite
mzip
```

You can also **mount** the floppy and use the normal commands, then unmount it. To **mount** a floppy you can type:

```
bash# mount /mnt/floppy
```

This will set the **/mnt/floppy/** directory to have the contents of the floppy root directory. Once you are done using the floppy unmount it by typing:

```
bash# umount /mnt/floppy
```

Note, the command is **umount** and not unmount.

8.3 Making CDs

CDs are a great way to store data because they are cheap and hold a lot of information compared to other media devices such as floppies and zip disks. A floppy disk holds about 1.4 MB. A zip disk holds about 80 times what a floppy holds. CDs hold about 7 times what a zip disk holds. Even though CDs hold the most they are the cheapest of the three. Read-Write CDs are now cheaper than floppy disks. In order to create a CD you first need a CD image, then you need to burn the CD. Many people do not know about the CD images because sometimes the images are created in memory while the CD is being burned.

8.3.1 CD Images

CD images hold all of the CD information including the file partition information, data, and information such as titles and copyright information. You can

create CD images with the **mkisofs** command. First create a directory with all of the files that you want on the CD. This directory will become the root directory on the CD. If your root directory was called **cdrom/** then you would use this command to create the CD image:

```
bash$ mkisofs -J -T -r -nobak -hide-joliet-trans-tbl -o image.iso cdrom/
```

If you want to put in information about the CD such as titles, copyright, and bibliography information you can create a file called **.mkisofsrc** in the directory that your **cdrom/** directory is in. Here is a sample **.mkisofsrc** file:

```
#The application identifier should describe the application that will be on
#the  disc.   There is space on the disc for 128 characters of information.
#May be overridden using the -A command line option.
APPI=My First CD
#
#The copyright information, often the name of a file on the disc containing
#the  copyright  notice.   There  is space in the disc for 37 characters of
#information.  May be overridden using the -copyright command line  option.
COPY=Copyright 2001, Bob
#
#The  abstract information, often the name of a file on the disc containing
#an abstract.  There is space in the disc for 37 characters of information.
#May be overridden using the -abstract command line option.
ABST=This is my first CD
#
#The  bibliographic  information, often the name of a file on the disc con-
#taining a bibliography.  There is space in the disc for 37  characters  of
#information.  May be overridden using the -bilio command line option.
BIBL=Bob
#
#This  should  describe  the  preparer of the CDROM, usually with a mailing
#address and phone number.  There is space on the disc for  128  characters
#of information.  May be overridden using the -p command line option.
PREP=Bob (bob@bob.org)
#
#This  should  describe  the publisher of the CDROM, usually with a mailing
#address and phone number.  There is space on the disc for  128  characters
#of information.  May be overridden using the -P command line option.
PUBL=Bob Publishing
#
#The  System  Identifier.   There is space on the disc for 32 characters of
#information.  May be overridden using the -sysid command line option.
SYSI=Linux Red Hat 7.1
#
#The Volume Identifier.  There is space on the disc for  32  characters  of
#information.  May be overridden using the -V command line option.
#This shows up under the icon in Windows 2000
VOLI=Bob CD #1
#
#The  Volume  Set  Name.   There is space on the disc for 128 characters of
#information.  May be overridden using the -volset command line option.
```

8.3.2 Burning CD Images

The easiest and probably most powerful way to burn CDs in Linux is with the **cdrecord** command. To make things easier there is a GUI program **xcdroast** that sets up and uses the command line options for **cdrecord** so that you do not need to know them. If you want to use the command line you can type this:

```
bash# cdrecord -v -eject image.iso
```

The **-v** option puts it in verbose mode so that you can see what is happening. The **-eject** causes the CD to eject after the burning has been completed.

8.3.3 Getting Images from CDs

It is easy to get an image from a CD with the **dd** command. Just type this:

```
bash# dd if=/dev/cdrom of=image.iso
1313572+0 records in
1313572+0 records out
```

This will create a file called **image.iso** that is ready to be burned onto another blank CD. It is probably a good to know if it is legal to copy a CD before you copy it.

8.4 Mounting and Unmounting Devices

Before you can read a device normally you need to mount the device. Some operating systems do not give you much of a choice as to where the device is mounted. In Linux you have lots of control. The file **/etc/fstab** tells you most of the default locations for devices to be mounted. If the device is listed in the **/etc/fstab** file you can easily mount and unmount. This is how you would mount and unmount a floppy disk and a cdrom respectively:

```
bash# mount /mnt/floppy
bash# umount /mnt/floppy
```

```
bash# mount /mnt/cdrom
bash# umount /mnt/cdrom
```

That looks pretty easy. You can also mount and unmount other things. A CD image file can be mounted with a line like this:

```
bash# mount -t iso9660 -o ro,loop=/dev/loop0 imagefile.iso directory/
```

When the CD image is mounted you can look around inside and make sure it looks right before you burn it.

8.5 Scheduling Tasks

Scheduled tasks are performed by the cron daemon. The **crontab** program lets you edit your cron file. Start the **crontab** editor by typing:

```
bash# crontab -e
```

Note that the default editor is **vi**. If you are not familiar with **vi** take a quick look at Section 4.5.2. Each line is a command to be run. The command is made up of five numbers and a command. The numbers are minutes, hours, days of week, days of month, and months. If you wanted the command **/sbin/reboot** to run every Sunday morning at 3:30 then you would have a line like this:

```
30 3 0 * * /sbin/reboot
```

When it does not matter what the value is you can put an asterisk there instead. This does not mean that minutes should be the asterisk if you do not care. You need to specify or you might have the program running every minute.

8.6 Copying to another System

You can copy your files from your system to another system many different ways. The important thing in backup is that you do not want to have to be there. There are three ways that I think you should consider and think about when you decide how to copy the files. NFS (network Filesystem), **scp** (secure copy), and **wget** (web get) all have advantages and disadvantages.

8.6.1 NFS Backup

With NFS you need to either mount another machines filesystem or export yours. If you mount the other machines filesystem then all you need to do is copy the files to the correct directory. If the backup machine is mounting your filesystem then it will need to do the backup copy.

The disadvantage is that the machines should be close together. You also leave the **sunrpc** port open for possible hack attempts. This is probably the easiest way to copy files though.

8.6.2 Scp Backup

You can setup an account on the backup machine to allow you to login with a ssh public key instead of a password. This method of authentication allows you to copy files and not worry about how to type the passwords in. To create the keys make sure you have a **.ssh/** directory in your home directory, then type:

```
bash$ ssh-keygen
Generating public/private rsa1 key pair.
Enter file in which to save the key (/home/bob/.ssh/identity):
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
Your identification has been saved in /home/bob/.ssh/identity.
Your public key has been saved in /home/bob/.ssh/identity.pub.
The key fingerprint is:
9c:32:e7:09:68:eb:49:21:45:30:49:f9:05:08:6f:7c bob@bob.org
```

This will create the ssh1 key. Next type:

```
bash$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/bob/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bob/.ssh/id_dsa.
Your public key has been saved in /home/bob/.ssh/id_dsa.pub.
The key fingerprint is:
55:cc:60:97:67:2f:52:0f:e7:da:76:fc:5f:b6:88:1c bob@bob.org
```

This will create the ssh2 key. Now take the files that end in **.pub** and copy them into files **authorized_keys** and **authorized_keys2** like this:

```
bash$ cp identity.pub authorized_keys
bash$ cp id_dsa.pub authorized_keys2
```

When you copy these files into the **.ssh/** directory of the account you want to log into, they should enable you to authenticate using the keys instead of a password. All you still need to do is create a shell script to copy the files using **scp**.

Here is an example **scp** command:

```
bash$ scp backupdata.tar.gz bob@backup.bob.org:backupdir/
```

8.6.3 Wget Backup

The idea behind **wget** is that you create a file and put it at a web location where the backup machine can download the file at a scheduled time. This can be difficult because the file might be in the creation process when the backup machine starts to download. This can get messy. Also, the file can be downloaded by other people who know the file is there. The major advantage is that the data does not require a login to copy and that removes some of the risks involved with passwordless logins and network filesystems. To download a file with **wget** use something like the following command:

```
bash$ wget http://bob.org/backupdata.tar.gz
```

Chapter 9

Security

9.1 Firewalls

Firewalls are good for security, but just because you have a firewall does not mean that your machine is secure. There are two main firewall tools that come installed with Red Hat Linux. You can setup and manage firewalls with the **ipchains** and **iptables** tools. Both of these were written by a guy named Rusty, who suggests you use the **iptables** firewalls. Since **ipchains** was written first there is a lot more GUI support for it. The only real difference between the two is how much control you have. Both of these firewall tools are good and should be tried out.

9.1.1 ipchains

ipchains is the easiest of the two to get working, infact your system is setup to work with **ipchains** by default. There are many tools to configure **ipchains**, I will start with the least powerful and move to the most powerful.

lokkit is the firewall tool that comes with the installation GUI. You can start it from the shell by typing:

```
bash# lokkit
```

You can also start it from the setup menu mentioned in Chapter 7 by typing:

```
bash# setup
```

After **lokkit** you have **firewall-config**. This one must be run from a GUI. That is the biggest drawback from this utility. If you are in X you can start this command from a terminal window by typing:

```
bash# firewall-config
```

The last and most powerful program is **emacs** (or **vi**). You can edit and create your own firewall rules by typing:

```
bash# emacs /etc/sysconfig/ipchains
```

In this **emacs** session you can custom create your own firewall rules. Here are some example rules:

```
-A input -s 0.0.0.0/0 -d 0.0.0.0/0 111:111 -p 6 -j REJECT -l
-A input -s 0.0.0.0/0 -d 0.0.0.0/0 67:69 -i eth0 -p 17 -j DENY
-A forward -s 10.0.0.0/8 -d 0.0.0.0/0 -j MASQ
```

In the above examples the **-A** input tells where the packet came from and what it is doing. **-s** is the source ip/mask. **-d** is the destination ip/mask. **111:111** is a port range. **-i eth0** refers to the network interface. **-p** refers to the protocal. Protocols are listed in the **/etc/protocol** file. **-j DENY** is the action that the rule is to take.

The first rule tells all computers that try to connect to tcp port 111 that the port is closed and logs the event. The second rule ignores all computers trying to connect to udp ports 67 to 69. The last rule forwards and masqurades all packets coming from the 10.x.x.x range to the servers own ip address and sends them on their way. In order for the third rule to work you also need to have routing enabled. This is talked about a little bit more in Chapter 10.

9.2 FTP and telnet

FTP and **telnet** are well known for their insecure nature. The passwords sent to these two servers can be easily sniffed and they fall easily to buffer overflow attacks. The reason that they are in such high use is that they are old legacy applications. Almost all of the old networking servers have both **ftp** and **telnet**. Many of the online web services allow you to publish pages or copy them to the server via **ftp**. The best thing to know is that if you are using **ftp** or **telnet**, you are likely to have problems. Now you have been warned.

9.3 Port Monitoring/Scanning

Port monitoring is a good way to know what the risks are. If you have a port monitoring program then you will probably know someone is planning an attack before it happens. Monitoring can be done by the firewall or by another server. **http://www.insecure.com/** provides a list of known security tools that can help you decide how to protect your system.

Port scanning is one of the most misunderstood security issues for new networking people. Port scanning is about the same as walking around a bank and looking at the cameras and checking out the security of the design. This is good for administrators who want to know about the security risks, but it does not look good if someone else does it to your server. Port scanning is when you try to connect to each port and see what happens. If there is a server running on the port you will probably get a response, but if the port is closed you will probably get a rejection packet from the server. Hackers and crackers scan a machine and look for ports like **ftp** and **telnet** that are know security risks. If these ports are open then they sometimes attempt an attack. The initial scan is perfectly legal, but what often follows is not legal.

9.4 Logs

Logs tell you important messages about what is going on with your machine. If you know what is happening then you are more likely to know when an attack is happening. This helps you prevent anything bad. The logs are found in the **/var/log/** directory. The logs that are probably the best to look at would be the **/var/log/messages** file and the **/var/log/secure** file. You can look at it by typing:

```
bash$ less /var/log/messages
```

Get familiar with the logs and know basically what to expect. If something looks strange then that might be an attackers sign.

9.4.1 What to look for?

When looking through the logs there are some things to look for. Buffer overflow attacks are usually full of extra long lines. Here is an exerpt from the **/var/log/httpd/error_log**:

```
10.9.8.7 - - [08/May/2002:15:03:09 -1000] "GET /default.ida?NNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858
%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u00
03%u8b00%u531b%u53ff%u0078%u0000%u00=a  HTTP/1.0" 400 326 "-" "-
"
10.13.15.17 - - [08/May/2002:23:17:15 -1000] "GET /default.ida?N
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6
858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%
u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a  HTTP/1.0" 400 326 "-"
 "-"
```

This is what the Code Red worm looks like. These should give you at least an idea of what a buffer overflow attack looks like. Sometimes attackers try to start someting on your machine take at this exerpt from the **/var/log/httpd/error_log**:

```
[Mon May 27 16:26:58 2002] [error] [client 10.10.10.10] File does
not exist: /var/www/html/msadc/..%5c../..%5c../..%5c/..^\../..^\.
./..^\../winnt/system32/cmd.exe
[Mon May 27 16:38:53 2002] [error] [client 10.11.12.13] File does
not exist: /var/www/html/robots.txt
```

The first line was a Nimda attack that should not really bother Linux users, but you should be aware of it. The second line is the request of a search engine that is looking for the **robots.txt** file to guide it in its mapping your web site. Search engines spiders are good if you want to be indexed into a search engine.

You should also consider logs such at the **lastlog** and the **/etc/passwd** file. To see when users logged in type:

```
bash# last | less
```

That will give you all of the logins for the current month. You should also check the password file to see if new accounts were created. Here is an example of a good account and a bad one:

```
root:x:0:0:root:/root:/bin/bash
root2:x:0:0:root2:/root:/bin/bash
```

If there is more than one account with 0 for the uid then you might have a problem. Attackers usually set up extra accounts with root privledges and use them to login. The root account is good, but you have to think, "Where did root2 come from?"

9.5 Passwords

Passwords need to be created well. Hackers prey on users who think up easy passwords to guess. Common crackable passwords include: dictionary words, dictionary words reversed, dictionary words with letters like o and i being replaced with digits like 0 and 1, dictionary words with stuff added on the end, keys in a row like asdf or 1qaz, etc. There are many password cracking programs that use dictionary files to test every password until they get a match. This could take a whole week, but that is just part of the fun for them.

A good password would be something that has mixed digits, lower case and upper case letters, and maybe a few symbols. To create such a password I find it good to create a phrase to remember and use the phrase for the letters. Here is a table of some such phrases:

Phrase	Password
To be or Not to be	2boN2b
I like to drive FAST cars	!l2dFc
The sign says Will work for food	TssWw4f

Chapter 10

Home Networking

There are a lot of reasons that someone would like to setup a home network. Some people do it because they want to have a private network where they can share services without the worry of hackers. Others do it because their Internet service provider charges extra for a second IP address. Whatever the reason, home networks can be a lot of fun to setup and administrate.

10.1 Private Networks

Many things bring privacy to a network. There the firewalls that bring a little bit of privacy, but usually people who have a private network set it up so that they have their own range of IP addresses. There are a few IP address ranges that are reserved for private use by the Internet Assigned Numbers Authority (IANA) (See RFC 1918). Here is a chart:

Address/Netmask	Comments
10.0.0.0/8	Free to use. (Old DARPA Range)
172.16.0.0/12	Free to use.
192.168.0.0/16	Free to use.

You can pick any of these ranges and use any of the addresses here as long as the addresses do not go out onto the Internet. The way that you can prevent their entering the Internet is to translate the addresses.

10.2 Network Address Translation

Common switches provide network address translation (NAT), but for a few dollars less you can get one without that service. If you cannot NAT the addresses with a switch you can do it with your computer. Install two network cards and use **ipchains** to translate the addresses. Here are some lines you could use to translate addresses listed above:

```
-A forward -s 10.0.0.0/8 -d 0.0.0.0/0 -j MASQ
```

```
-A forward -s 172.16.0.0/12 -d 0.0.0.0/0 -j MASQ
-A forward -s 192.168.0.0/16 -d 0.0.0.0/0 -j MASQ
```

Just put these lines in the **/etc/sysconfig/ipchains** file. The next time you start **ipchains** the rules will be in effect. This is how you would restart ipchains:

```
bash# /etc/rc.d/init.d/ipchains stop
Flushing all chains:                                       [  OK  ]
Removing user defined chains:                              [  OK  ]
Resetting built-in chains to the default ACCEPT policy:    [  OK  ]
bash# /etc/rc.d/init.d/ipchains start
Flushing all current rules and user defined chains:        [  OK  ]
Clearing all current rules and user defined chains:        [  OK  ]
Applying ipchains firewall rules:                          [  OK  ]
```

10.3 Enabling Routing

On Red Hat Linux machines and many other Linux Distributions the network packet routing is turned off by default. This is more of a safety issue than anything else. If the computer does not route packets then there is less risk involved with routing attacks. To turn the routing on you need to set the boolean kernel perameter called **ip_forward** to true. This can be done using the following command:

```
bash# echo 1 > /proc/sys/net/ipv4/ip_forward
```

While this does work to turn the routing on, it is best to make sure you know how to keep it on. If you put the above line in one of the startup files then you can be sure that routing will be enabled every time you turn the computer on. The following command would do this:

```
bash# echo "echo 1 > /proc/sys/net/ipv4/ip_forward" >> \
      /etc/rc.d/rc.local
```

If you mess up and mistype the above line you could do a lot of damage, so make sure you type it right. If you are not willing to mess up badly then you can use an editor instead to add the first line to the end of the **/etc/rc.d/rc.local** file.

Part II

Servers

Chapter 11

MySQL Server

MySQL is is a SQL-based relational database server. MySQL can be installed by selecting the SQL server package during installation. For this next Chapter we will assume you selected the everything package during the Red Hat Linux installation. Although MySQL is installed it is not activated until you start it. MySQL is useful if you are thinking about having a Web site that uses many of the features of a E-commerce web site out in industry.

11.1 Setting up mysqld

In order to start MySQL all you have to do is start the server:

```
bash# /etc/rc.d/init.d/mysqld start
```

When the server is running you will probably want to create the symbolic links so that MySQL server starts up every time that the computer reboots. To do this type:

```
bash# chkconfig mysqld on
```

Once this is taken care of you can start up a MySQL client called **mysql**. Type:

```
bash# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.23.36

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql>
```

This will take you to the MySQL prompt where you can start creating databases, tables, and tuples as well adding rights for users. The first thing to do is probably to figure out what is in the MySQL database right now.

```
mysql> show databases;
+------------+
```

```
| Database   |
+------------+
| mysql      |
| test       |
+------------+
2 rows in set (0.08 sec)
```

These two databases are the default databases. The **mysql** database holds information about the users and their passwords. To see this database you must first use it:

```
mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Now you are in the **mysql** database. To see the tables you can type:

```
mysql> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.00 sec)
```

You can see all of the users on your machine by typing:

```
mysql> select Host, User from user;
+----------------------+--------+
| Host                 | User   |
+----------------------+--------+
| localhost            | root   |
+----------------------+--------+
1 rows in set (0.00 sec)
```

You can add additional users by using the grant command. Here is how you would give Bob whose login is a "bob" access to use the test database:

```
mysql> grant all on test.* to bob@localhost;
Query OK, 0 rows affected (0.01 sec)
```

It says 0 row affected but really a few rows were created in various tables in the mysql database. Once you start to create users they can start to also use mysql. To leave MySQL type:

```
mysql> exit
Bye
```

Now Bob could log in by typing the command:

```
bash$ mysql -h localhost
```

Ideally you create another account for bob@SERVER where SERVER is your computers hostname. If you do this then Bob can use the computer's hostname as the switch instead of localhost.

11.2 Databases, tables, and tuples

Lets say that we want to create a database for Bob to use for his own stuff and give him rights to use this database. This is quite easy to do:

```
mysql> create database bob;
Query OK, 1 row affected (0.02 sec)

mysql> grant all on bob.* to bob@localhost;
Query OK, 0 rows affected (0.01 sec)

mysql> use bob
Database changes
```

That was easy. Let's create a table for Bob to use:

```
mysql> create table messages (
    -> number int(6),
    -> sender varchar(250),
    -> text varchar(250)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

We can now look at the description of this table:

```
mysql> describe messages;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| number | int(6)       | YES  |     | NULL    |       |
| sender | varchar(250) | YES  |     | NULL    |       |
| text   | varchar(250) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Okay, that looks good. Now to enter a tuple of data. (Tuples are often called records.)

```
mysql> INSERT INTO messages VALUES ('1', 'admin', 'Hello Bob, how are you?');
Query OK, 1 row affected (0.03 sec)
```

Now we can look at the table to see if the record is there:

```
mysql> select * from messages;
+--------+--------+-------------------------+
| number | sender | text                    |
+--------+--------+-------------------------+
|      1 | admin  | Hello Bob, how are you? |
+--------+--------+-------------------------+
1 row in set (0.00 sec)
```

It looks like the tuple is there.

11.3 Perl DBI

Now we want to see if we can get Perl to work with MySQL. Perl uses a program called DBI to interface with the MySQL database server. You can download and install two packages from **http://www.mysql.com/** to use DBI. Look for DBI on the downloads page. The two packages are DBI and Msql-Mysql-modules. These two files will have version numbers and other things like that. Download them and let's get to installing them.

11.3.1 Installing DBI

In order to install DBI you first need to uncompress the file. At the writing of this book the current stable version of DBI was 1.18. This is how we can expand the file:

```
bash# tar xfz DBI-1.18.tar.gz
```

This should create a directory called DBI-1.18. Lets go into the directory and start the compile and then install the package:

```
bash# cd DBI-1.18/
bash# perl Makefile.PL
bash# make
bash# make install
```

There will be other stuff that comes on the screen, but if none of it looks like error messages then you are probably doing everything correctly. Now, let's go back to the directory below us and expand the next one.

11.3.2 Installing Msql-Mysql-modules

You must install DBI before you can install the Msql-Mysql-modules package. The package I will use is version 1.2216.

```
bash# tar xfz Msql-Mysql-modules-1.2216.tar.gz
```

That created a directory called **Msql-Mysql-modules-1.2216/**. Now to compile and install this one:

```
bash# cd Msql-Mysql-modules-1.2216
bash# perl Makefile.PL
```

This will give you a list of options. Select 1 MySQL only. Then press **Enter** to select the default for the rest of the options. Let's continue:

```
bash# make
bash# make install
```

Now DBI should be up and working. Now comes the fun part.

11.3.3 Testing Perl DBI

You can create a Perl program that will read messages from the table that we created. Here is some example program called **dbitest.pl** that will do just this:

```
#!/usr/bin/perl

use DBI;

$dsn = "DBI:mysql:bob:localhost"; #data source name
$user_name = "bob"; # user name
$password = ""; # passwords are either blank or existing

$dbh = DBI->connect($dsn, $user_name, $password, { RaiseError => 1});

$count = 0;
$infoget = "SELECT * FROM messages";
$sth = $dbh->prepare ($infoget);
$sth->execute();
while(@info = $sth->fetchrow_array()) {
    ($num, $sender, $message) = @info;
    print "$num ($sender) - $message\n";
    $count++;
}

print "$count Message(s)\n";

exit;
```

To run and test this program first change the permissions to 755 (as explained in Section 4.6) then run the program:

```
bash$ ./dbitest.pl
1 (admin) - Hello Bob, how are you?
1 Message(s)
```

11.4 MySQL and CGI

Once you have DBI and Perl working together you can expand your Perl programs so that they work as web driven CGI programs. With this you can create online surveys, E-commerce sites and much, much more. We will talk about setting up a web server and using DBI in Perl programs in Chapter 12.

Chapter 12

Web Server

If you installed everything when you did the computer installation you should have the Apache web server installed. The Apache Web Server is one of the popular servers out on the Internet. It is very powerful and fairly easy to install and use.

12.1 Setting up Apache

To setup your Apache server all you really need to do is turn it on. To turn the server on you can run the command:

```
bash# /etc/rc.d/init.d/httpd start
```

This will start the server running, but the server will not be running when you reboot unless you run the command:

```
bash# chkconfig httpd on
```

This creates symbolic links in the /etc/rc.d/ directory tree that tell the computer which servers to start when it boots up. If you would like to know what run levels the server is running at you can type the following command:

```
bash# chkconfig --list httpd
httpd           0:off 1:off 2:off 3:on 4:on 5:on 6:off
```

This tells you that as long as the computer is in run level 3, 4, or 5 the web server should be running. Most servers run in level 3, but some run in level 5. Either one should be fine for what we want.

12.2 Simple HTML

Once the server is running you can immediately start to see web pages in the **/var/www/html/** directory. If you want to you can create a simple HTML file and test out your web server. This is what the code for a simple html called **simple.html** could look like:

```
<html>
<head>
<title>Simple HTML Title</title>
</head>
<body bgcolor=white>
<h1>My Simple HTML Page</h1>
What do you think?
</body>
</html>
```

Put this file in your **/var/www/html/** directory then using a browser open the URL **http://localhost/simple.html** and see this html file. If you are not seeing this HTML it is possible that the permissions are not set right. While in the **/var/www/html** directory type the command:

```
bash# chmod 644 simple.html
```

This will change the rights of the HTML file so that it can be seen on the Internet.

12.3 CGI Scripting

CGI is short for Common Gateway Interface. The idea behind CGI is that you have a program that runs and generates a web page instead of an unchanging text html file. CGI is also able to read information from its environment, a query string, and from standard input. Through these three input methods the program can know a lot of information about the type of page that it is supposed to create. When you use Apache Web Server you have the **/var/www/cgi-bin/** directory in which you can place and run CGI scripts by default. The scripts can then be seen from a browser on your computer. If a script were placed at **/var/www/cgi-bin/script.cgi** you could view it by typing the location **http://localhost/cgi-bin/script.cgi** in the URL location. If you cannot get a script to run you might want to make sure the script has the correct permissions. Change the CGI script to 755:

```
bash# chmod 755 /var/www/cgi-bin/script.cgi
```

Unlike the HTML files the CGI files have more restrictions. CGI need to have the execute bit set, but they cannot have the world write bit set. Because of this the numbers 777 and 766 are both invalid. For Perl or other scripting languages they need to be world readable in order to execute. Because of this numbers like 751 and 711 would only work for binary CGI programs. For Perl 755 is probably the best.

12.3.1 Environment

Every program has an environment in which it runs. In Perl the environment is stored in a hash variable called %ENV. In order to print the environment from a CGI program you could create the following perl CGI program **env.cgi**:

```
#!/usr/bin/perl
```

```
print "Content-type: text/html\n\n";

print "<html>\n";
print "<head>\n";
print "<title>Environment</title>\n";
print "</head>\n";
print "<body bgcolor=ffffff>\n";

print "<h1>Environment</h1>\n";
foreach $x (keys(%ENV)) {print "$x=$ENV{$x}<br>\n";}

print "</body>\n";
print "</html>\n";

exit;
```

You could then run the program and see the environment that the program runs in.

12.3.2 Standard Input

In the environment you have a variable $ENV{'CONTENT_LENGTH'} that tells you the number of bytes of standard input that your program can read in. Standard input is generated from an HTML form that used the POST method. If you were interested in seeing the results of the standard input in a readable form you could use the following program **stdin.cgi**:

```
#!/usr/bin/perl

read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
        ($name, $value) = split(/=/, $pair);
        $name =~ tr/+/ /;
        $name =~ s/%([a-f0-9][a-f0-9])/pack("C", hex($1))/eig;
        $value =~ tr/+/ /;
        $value =~ s/%([a-f0-9][a-f0-9])/pack("C", hex($1))/eig;

        push (@slist, $name);
        $sinput{$name} = $value;
}

print "Content-type: text/html\n\n";

print "<html>\n";
print "<head>\n";
print "<title>Standard Input</title>\n";
print "</head>\n";
print "<body bgcolor=ffffff>\n";

if (@slist) {
    print "<h1>STDIN</h1>\n";
```

```
    foreach $x (@slist) {
print "$x=$sinput{$x}<br>\n";
    }
}

print "</body>\n";
print "</html>\n";

exit;
```

This program is only good if you have an HTML form that points to this file. You could create an html file called **/var/www/html/stdin.html** that looked like this:

```
<html>
<head>
<title>STDIN Form</title>
</head>
<body bgcolor=ffffff>
<h1>STDIN Form</h1>
<form action=/cgi-bin/stdin.cgi method=post>
<input type=text name=text1>
<input type=text name=text2>
<input type=text name=text3>
<input type=submit>
</form>
</body>
</html>
```

From this HTML you could then see what you had typed into the three text boxes when the CGI program ran.

12.3.3 Query String

The query string is basically the same as the standard input except the query string cannot hold as much information as standard input can, the query string becomes part of the URL, and you use the GET method instead of POST. This is good when you want to bookmark a CGI page, but it is bad when you want to transmit information securely. Not that information transmitted in the query string is transmitted in a different way then standard input, but if it is on the query string the user can see the information. While the program has to read in standard input, the query string is contained in the environment. To see the query string you can create the file **/var/www/cgi-bin/query.cgi**:

```
#!/usr/bin/perl

$buffer = $ENV{'QUERY_STRING'};
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
        ($name, $value) = split(/=/, $pair);
        $name =~ tr/+/ /;
        $name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
        $value =~ tr/+/ /;
```

```
        $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

        push (@qlist, $name);
        $qinput{$name} = $value;
}

print "Content-type: text/html\n\n";

print "<html>\n";
print "<head>\n";
print "<title>Query String</title>\n";
print "</head>\n";
print "<body bgcolor=ffffff>\n";

if (@qlist) {
    print "<h1>Query String</h1>\n";
    foreach $x (@qlist) {
print "$x=$qinput{$x}<br>\n";
    }
}

print "</body>\n";
print "</html>\n";
exit;
```

To send information to the query string you can change the HTML for standard input so that the form line says this:

```
<form action=/cgi-bin/query.cgi method=get>
```

You can also type the URL directly. Try some of the following:

```
http://localhost/cgi-bin/query.cgi?a=1&b=2&c=3
http://localhost/cgi-bin/query.cgi?search=dog+cat+mouse
http://localhost/cgi-bin/query.cgi?text1=apple&text2=banana
```

12.4 Cookies

Cookies are information packets that are stored locally on your computer. Cookies are also passed to the CGI program as part of the environment like the query string. On the Internet cookies are often used to remember who you are and what you like. Some companies will work together to add information about you and put them in cookies. Later those companies will put up advertisements based on the things in which you seem to be interested. If you have the right interests and go to the right sites this can help you. If you mess up and accidentally go to the wrong site who knows when all of the junk advertisements will end.

12.4.1 Creating Cookies

You can create a cookie very easily. Here is a program (**cookie.cgi**) that creates a cookie that counts how many times you have visited the page. First you need

to know the name of your server. Replace SERVER with your hostname:

```
#!/usr/bin/perl

$cookie = $ENV{'HTTP_COOKIE'};
if ($cookie) {
    if ($cookie =~ /count=([^;]*)/) {$count = $1;}
}
if ($count) {
    $count++;
} else {
    $count = 1;
}

print "Content-type: text/html\n";
print "Set-Cookie: count=$count; path=/; domain=SERVER\n\n";

print "<html>\n";
print "<head>\n";
print "<title>Cookie Counter</title>\n";
print "</head>\n";
print "<body bgcolor=ffffff>\n";

print "You have visited $count time(s).<br>\n";

print "</body>\n";
print "</html>\n";

exit;
```

You can then look at the page at **http://SERVER/cgi-bin/cookie.cgi** where SERVER is your computers hostname. Repeatedly pressing the Reload button should give you a different number each time. If the number is constantly 1 then there are two likely possibilities. Either you did not change the word SERVER to your server's hostname or the browser is not accepting cookies.

12.5 Database

You can add the database element to a CGI program quite easily. Make sure that you have completed the MySQL installation in Chapter 11 before you try using DBI in CGI programs. Let's now take the example Perl program that used DBI and turn it into a CGI program **/var/www/cgi-bin/dbitest.cgi**:

```
#!/usr/bin/perl

use DBI;

$dsn = "DBI:mysql:bob:localhost"; #data source name
$user_name = "bob"; # user name
$password = ""; # passwords are either blank or existing

$dbh = DBI->connect($dsn, $user_name, $password, { RaiseError => 1});
```

```
print "Content-type: text/html\n\n";

print "<html>\n";
print "<head>\n";
print "<title>dbitest.cgi</title>\n";
print "</head>\n";
print "<body bgcolor=ffffff>\n";
print "<h1>dbitest.cgi</h1>\n";
print "<ul>\n";

$count = 0;
$infoget = "SELECT * FROM messages";
$sth = $dbh->prepare ($infoget);
$sth->execute();
while(@info = $sth->fetchrow_array()) {
    ($num, $sender, $message) = @info;
    print "<li>$num ($sender) - $message\n";
    $count++;
}

print "</ul>\n";
print "$count Message(s)<br>\n";

print "</body>\n";
print "</html>\n";

exit;
```

Once you have the program there and permissions set, go and take a look at it. With DBI programs, since they are Perl programs you can debug them by running them from the shell most of the time.

12.6 Configuring Apache

There are tools for configuring Apache, but modifying the configuration file gives you a lot more control over what is happening. If you would like to take a look at the configuration tool, start:

```
bash# apacheconf
```

If you would like to have more control, change into the **/etc/httpd/conf/** directory and take a look around. There should a few files that are interesting, but only one file that you need to edit. The file to edit is **/etc/httpd/conf/httpd.conf**. Open this file with an editor and let's begin.

12.6.1 Enabling User CGI

In order to allow users to use CGI in their **public_html/** directories there are a few things that need to be changed from the default configuration. First let's open up the **/etc/httpd/conf/httpd.conf** file and see what the cgi-script handler looks like before and after. Find the line that looks like this:

```
#AddHandler cgi-script .cgi
```

Once you have the line, remove the pound sign (#) and decide which file extensions will be treated as CGI programs. Let's let the **.pl** extension also mean that the file is a CGI program. This is how the new line should look:

```
AddHandler cgi-script .cgi .pl
```

Okay, now that is done. Now we have to give the users rights to use CGI. The user HTML files are stored in a directory called **public_html/**. Find the place where the UserDir directories configuration is. Here is some example code you should find:

```
# Control access to UserDir directories.  The following is an example
# for a site where these directories are restricted to read-only.
#
#<Directory /home/*/public_html>
#    AllowOverride FileInfo AuthConfig Limit
#    Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
#    <Limit GET POST OPTIONS PROPFIND>
#        Order allow,deny
#        Allow from all
#    </Limit>
#    <Limit PUT DELETE PATCH PROPPATCH MKCOL COPY MOVE LOCK UNLOCK>
#        Order deny,allow
#        Deny from all
#    </Limit>
#</Directory>
```

This is all commented and is not used, but it shows you basically what you need to do to configure user directories. Add code like this after the commented section:

```
<Directory /home/*/public_html>
    AllowOverride None
    Options ExecCGI Indexes Includes MultiViews SymLinksIfOwnerMatch
    Order allow,deny
    Allow from all
</Directory>
```

Once you finish the changes to the configuration file you will have to restart the server:

```
bash# /etc/rc.d/init.d/httpd restart
```

Now everything should be working. To test the page on Bob's website let's first change the permissions to his home directory, then create a **public_html/** directory that he has rights to work in:

```
bash# chmod 711 /home/bob/
bash# mkdir /home/bob/public_html
bash# chown bob /home/bob/public_html
```

Once this is completed you can copy the CGI file **/var/www/cgi-bin/dbitest.cgi** to Bob's **public_html/** directory and view it on the server at the URL: **http://localhost/ bob/dbitest.cgi** with a browser.

12.6.2 Default Error Pages

When you surf the Web there are web pages that pop up every once in a while that tell you that the page you are trying to see is no longer available. Sometimes that is because the web page has changed, and sometimes it is because the web page is not there and the server has decided to display another page instead. This is an easy thing to setup yourself. Open up the **/etc/httpd/conf/httpd.conf** configuration file again and search for the place that looks like this:

```
# Customizable error response (Apache style)
#  these come in three flavors
#
#    1) plain text
#ErrorDocument 500 "The server made a boo boo.
#  n.b.  the (") marks it as text, it does not get output
#
#    2) local redirects
#ErrorDocument 404 /missing.html
#  to redirect to local URL /missing.html
#ErrorDocument 404 /cgi-bin/missing_handler.pl
#  N.B.: You can redirect to a script or a document using server-side-includes.
#
#    3) external redirects
#ErrorDocument 402 http://some.other_server.com/subscription_info.html
#  N.B.: Many of the environment variables associated with the original
#  request will *not* be available to such a script.
```

Everything here is commented out, but there are four examples of what you can do. Example 1 shows how you could have the server display a plain text page with what ever text you wanted. Example 2 shows two types of redirects. You can either redirect a URL to a local HTML or local CGI program. In Example 3 you have external redirects. This is if the server changed and you wanted to page to automatically redirect to the new server. There are other reasons you could use it to redirect to the outside also. Here are some example error page configurations:

```
ErrorDocument 403 /error403.html
ErrorDocument 404 /error404.html
ErrorDocument 405 /error405.html
ErrorDocument 500 /error500.html
```

It is quite possible that you will never get some of these errors, but it is still fun to create the web pages. If you would like a complete list of the errors that can or might happen in the future you can check out the Request For Comments (RFC) document number **rfc2616**. You can probably find a copy of the document anywhere on the Internet by typing **rfc2616** into a search engine, but there was a copy at this address when I wrote this:

http://www.w3.org/Protocols/rfc2616/rfc2616.html

Chapter 13

Mail Server

Sendmail is a nice server for sending and receiving mail. **sendmail** has a daemon that sits and watches port 25 and listens for mail. Usually **sendmail** is on by default, but it is not accepting mail from outside the computer. Many of the computer processes send mail to the root account by default on a frequent basis.

13.1 Setting up Sendmail

You can either setup sendmail by modifying the **/etc/sendmail.cf** configuration file, or by using the mailconf command. Start up **linuxconf** and see if the **mailconf** module is turned on.

```
bash# linuxconf
```

From Linuxconf select the Control tab in GUI mode or scroll down to the Control section in text mode. Select Configure Linuxconf Modules. A lot of these look interesting, but the one we want is the **mailconf** module. Make sure that one is selected, then exit linuxconf.

```
bash# mailconf
```

Next start **mailconf**. Configure your machine to present mail as from itself. This is either in the basic information section or the configure section. That is about all you need to do to configure sendmail. Now exit and type this command:

```
bash# /etc/rc.d/init.d/sendmail restart
```

If that command fails at shutting down mail, then you might need to see if the mail server is starting at startup. You will most likely get one of the two following situations:

```
bash# chkconfig --list sendmail
sendmail        0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

Which is a good sign. Or:

```
bash# chkconfig --list sendmail
sendmail        0:off 1:off 2:off 3:off 4:off 5:off 6:off
```

Which is a also a good sign. If you get the second one you can use **chkconfig** to get your server to run regularly:

```
bash# chkconfig sendmail on
```

This should do the trick.

13.2 Aliases

Email Aliases are stored in the **/etc/aliases** file. You can modify the file by hand or you can use **mailconf** to do it. By hand is pretty easy. Here is a sample section of an aliases file:

```
# Basic system aliases -- these MUST be present.
mailer-daemon: postmaster
postmaster: root

# General redirections for pseudo accounts.
bin: root
daemon: root
adm: root
lp: root
sync: root
shutdown: root
halt: root
mail: root
```

You can add other aliases to this file by simply inserting a line with the name: redirectedname format. To create an address john that emailed a user bob you could add the line:

```
john: bob
```

You can also have multiple addresses on the same line:

```
admin: dave,bob,sara
```

Not only can you direct mail to another user, but you can direct mail to a forward list or a program. This is how you would do a forwarding list:

```
admin: :include:/etc/admin.list
```

To forward to a file use a line a bit like this:

```
admin: "| /root/admin.pl"
```

All of this can also be done easily with **mailconf** using the user alias option. With the aliases that forward email to programs and lists you need to be careful. A list might have an email that bounces. If it bounces then you might get the bounced letters forwarded to everyone on the list. That could cause a crazy email storm. With a program you need to make sure it is written correctly because the program with run with more rights then an ordinary user. This could get dangerous if you mess up. It is fun to play with and basically harmless on a small server.

13.3 Spam

Everyone seems to dislike spam messages. It gets annoying when you have to sit and delete message after message until you can finally read your email. You can setup the mail server to block or reject messages by using mailconf. Under Anti-spam filters you have the option to reject senders. For each site that sends spam mail, add an entry that rejects mail from that site.

You can also go into the **/etc/mail/** directory and edit the deny file so that there is a line that looks something like this:

```
spammer.com Sorry, we are not currently accepting spam.
```

This will send a message back to the site when they connect that says: "Sorry, we are not currently accepting spam." After editing the file compile it into a database file by running the command:

```
bash# make
```

If nothing happens, it is possible that the deny file is not listed in the **Makefile**. Make sure the all: line looks like this:

```
all: ${POSSIBLE} virtusertable.db access.db domaintable.db \
     mailertable.db deny.db
```

Each of the database files can be generated this way. If you modify the **/etc/mail/sendmail.mc** file you can generate the new **/etc/sendmail.cf** file by running this command:

```
bash# m4 /etc/mail/sendmail.mc > /etc/sendmail.cf
```

13.4 .forward and .procmailrc Files

You can direct your mail to go to files, programs, or filter in other ways using a **.forward** or **.procmailrc** file in your home directory. The **.forward** files are discussed in Section 5.2.1 and the **.procmailrc** files are discussed in Section 5.2.2.

Chapter 14

NFS/NIS Servers

Network Filesystems (NFS) and Network Information Servers (NIS) work well together. Using these two you can setup very nice networks. I have discovered a few bugs that allow you to get into other accounts sent out with the NIS information, but overall it is very nice. NFS is where you export a file system and the other machine mounts your filesystem as if it were on the machine locally. NIS allows sharing of passwords among a trusted group of computers in a network.

14.1 Exporting Directories

NFS allows you to export a directory onto the network. This can be done with the **/etc/exports** file by creating a line that looks like this:

```
/disk 192.168.1.0/24(rw)
```

This exports the **/disk** directory to the whole **192.168.1.0/24** class C network. Any machine on that network may then mount the filesystem and both read and write to it. You could also pick to export to just one machine if you wanted to with a line like this:

```
/disk 192.168.1.1(rw)
```

You can also export to machines by name.

14.2 Mounting Directories

Once a file system is exported, you may mount it by first creating a directory to mount it on, then by adding a line to your **/etc/fstab** file. Let's first create a directory called **/nfs**.

```
bash# mkdir /nfs
```

Now that there is a directory there, let's modify the **/etc/fstab** file and add a line like this:

```
nfs.bob.org:/disk /nfs nfs bg,hard,rsize=8192,wsize=8192
```

You would need to change the server name from **nfs.bob.org** to whatever your server name is. Once this line is in there, you are ready to mount the directory.

```
bash# mount /nfs
```

Take a look around the directory and make sure everything looks right. If the PIDs in the NFS file system do not match the ones on your machine then you will get numbers instead of names. This can be solved by just using the same PIDs and accounts. This is available by using NIS.

14.3 Setting up a NIS Master Server

In this section we will setup an NIS server on a machine named **nis.bob.org** in our network. NIS servers are nice because they allow many machines on a network to share passwords and other information. This is very nice in a lab setting where users of the lab may want to log into different machines without needing to remember multiple passwords.

In order to setup and run an NIS server you need to have **portmap** running. Since **portmap** is on by default you should not have to do anything, but just incase it is off you should check it. You can test this by running this command:

```
bash# rpcinfo -u localhost ypserv
program 100004 version 1 ready and waiting
program 100004 version 2 ready and waiting
```

If you get something other than the above response then you will need to start **portmap**.

Once you have **portmap** running you will need to decide a domainname for the NIS information to be sent on. Let's pick the name bob.org for our domainname. Set the domainname with the **ypdomainname** command like this:

```
bash# ypdomainname bob.org
```

Now let's initialize the master server. For this example we will not have any secondary servers:

```
bash# /usr/lib/yp/ypinit -m

At this point, we have to construct a list of the hosts which will run NIS
servers.  nis.bob.org is in the list of NIS server hosts.  Please continue to add
the names for the other hosts, one per line.  When you are done with the
list, type a <control D>.
next host to add:  nis.bob.org
next host to add:
The current list of NIS servers looks like this:

nis.bob.org
```

```
Is this correct?  [y/n: y]
We need some  minutes to build the databases...
Building /var/yp/bob.org/ypservers...
Running /var/yp/Makefile...
gmake[1]: Entering directory '/var/yp/bob.org'
Updating passwd.byname...
Updating passwd.byuid...
Updating group.byname...
Updating group.bygid...
Updating hosts.byname...
Updating hosts.byaddr...
Updating rpc.byname...
Updating rpc.bynumber...
Updating services.byname...
Updating services.byservicename...
Updating netid.byname...
Updating protocols.bynumber...
Updating protocols.byname...
Updating mail.aliases...
gmake[1]: Leaving directory '/var/yp/bob.org'
```

At the prompt we want to just press **Ctrl-d** and then confirm the list. As you can see there were multiple files worked with. Most of the important stuff happens in the **/var/yp/** directory. Now start the **ypserv** server:

```
bash# /etc/rc.d/init.d/ypserv start
```

If you ever change passwords or create new accounts then you will need to make sure the new data gets loaded into the NIS servers. You can do this by typing the following lines:

```
bash# cd /var/yp/
bash# make
```

That should take care of your NIS server. Now for the client.

14.4 Setting up NIS Clients

Setting up the NIS client is probably the easiest part of the NIS process. When you install Red Hat Linux you are given the choice in the authentication section about the authentication method. If you wanted to setup a client to use the **nis.bob.org** server with the bob.org domainname you would input those choices in the domain and server fields. Everything should work form there.

If your machine is running as an NIS client you will not be able to use the full range of PIDs. There are MINUID and MINGID variables set in the Makefile of the server that define the range of PIDs. The default is set to 500 which will allow the client to use the first 500 PIDs for client use. After the 500 all of the accounts will come from the NIS server. These files kind of get joined in memory.

If you would like to see the NIS password file as it is sent type this command on the client machine:

```
bash# ypcat passwd
```

Chapter 15

DNS Server

The DNS translates the hostnames into IP addresses and IP addresses into hostnames. This is done on Linux systems with the named Name Server Daemon. Setting up and running a DNS is not a very difficult thing to do.

15.1 Setting up named

The DNS requires a lot of files in order to run everything properly. Once you have all of the files in place you can run the command:

```
bash# /etc/rc.d/init.d/named start
```

The first thing you need to in order to set up named is to edit the configuration file. The configuration file is found at **/etc/named.conf**. This is basically what the file looks like:

```
// generated by named-bootconf.pl

options {
directory "/var/named";
/*
 * If there is a firewall between you and nameservers you want
 * to talk to, you might need to uncomment the query-source
 * directive below.  Previous versions of BIND always asked
 * questions using port 53, but BIND 8.1 uses an unprivileged
 * port by default.
 */
// query-source address * port 53;
};

//
// a caching only nameserver config
//
zone "." IN {
type hint;
file "named.ca";
};
```

This file works nice, but lets create a new file to replace this one. Let's create the domain name **bob.org** and assume that we have the IP addresses from **1.2.3.0** to **1.2.3.255**. This will be the server **1.2.3.2**. Lets also assume that another DNS at the address **2.3.4.5** is going to be getting a copy of our DNS. This will only be one of two DNSs that we will create. Our two DNSs will be **1.2.3.2** and **1.2.3.3**. Change the file to say this instead:

```
options {
directory "/var/named";
allow-transfer{
2.3.4.5;
1.2.3.2;
1.2.3.3;
};
};
zone "." {
type hint;
file "named.ca";
};
zone "0.0.127.in-addr.arpa"{
type master;
file "named.local";
};
zone "bob.org"{
type master;
file "db.bob";
};
zone "3.2.1.in-addr.arpa" {
        type master;
        file "db.1.2.3";
};
```

Notice that when we declare a zone we reverse the numbers in the IP address and drop the last number that became the first. We have to allow transfers for **2.3.4.5** because that is the DNS that will have our information and publish it to the rest of the world. We are allowing transfers to ourself and the other backup DNS server at **1.2.3.3**. The other DNS as **1.2.3.3** would have a file very similar that would look like this:

```
options {
directory "/var/named";
allow-transfer{
2.3.4.5;
1.2.3.2;
1.2.3.3;
};
};
zone "." {
type hint;
file "named.ca";
};
zone "0.0.127.in-addr.arpa"{
type master;
file "named.local";
```

```
};
zone "bob.org"{
type slave;
file "db.bob";
masters{
1.2.3.2;
};
};
zone "3.2.1.in-addr.arpa" {
        type slave;
        file "db.1.2.3";
masters{
1.2.3.2;
};
};
```

About all that needs to be changed is the line that says "type master" so that it instead a slave and there needs to be a master section that says where the master server is located.

15.2 Local Zones

Create a file called **/var/named/named.local** on both the main DNS server (**1.2.3.2**) and the secondary server (**1.2.3.3**) and put this data in it:

```
$TTL 86400
@       IN      SOA     localhost. root.localhost.  (
                                        1997022700 ; Serial
                                        28800      ; Refresh
                                        14400      ; Retry
                                        3600000    ; Expire
                                        86400 )    ; Minimum
              IN      NS      localhost.

1       IN      PTR     localhost.
```

Now all you have to do is create the forward zone and the reverse zones.

15.2.1 Forward Zone

The forward zones are the ones that are used to translate the hostnames into IP addresses. Here is an example file for the **bob.org** zone:

```
@ IN NS dns.bob.org.
@ IN NS dns2.bob.org..
$ORIGIN edu.
bob IN SOA bob.org. root.bob.org. (
2002121301 ; serial
10800 ; refresh
3600 ; retry
604800 ; expire
```

```
86400 ; default_ttl
)
bob IN NS dns.bob.org.
bob IN NS dns2.bob.org.
$ORIGIN bob.org.
localhost IN A 127.0.0.1
gateway IN A 1.2.3.1
dns IN A 1.2.3.2
dns2 IN A 1.2.3.3
bob.org. IN A 1.2.3.4
www IN CNAME bob.org.
```

This will set up **bob.org** so that there are four machines with hostnames. The gateway, which is probably a switch is sitting at **1.2.3.1**. The name servers are at **1.2.3.2** and **1.2.3.3**. Their names are **dns.bob.org** and **dns2.bob.org**. The last machine is the **bob.org** web server. This machine is at **1.2.3.4** and has two names. The real name or the CNAME is **bob.org**. It also has an alias **www.bob.org** which points to the machine with the CNAME **bob.org**.

15.2.2 Reverse Zone

The reverse zones give you a hostname for every IP address assigned to that zone. Bob has decided that all IP address that are not currently used will have a name like **bob123.bob.org**. This is what the reverse zone would look like:

```
@ IN NS dns.bob.org.
$ORIGIN 2.1.IN-ADDR.ARPA.
3 IN SOA bob.org. root.bob.org. (
2001121401 ; serial
10800 ; refresh
3600 ; retry
604800 ; expire
86400 ; default_ttl
)
3 IN NS dns.bob.org.
3 IN NS dns2.bob.org.
$ORIGIN 3.2.1.IN-ADDR.ARPA.
0 IN PTR network.bob.org.
1 IN PTR gateway.bob.org.
2 IN PTR dns.bob.org.
3 IN PTR dns2.bob.org.
4 IN PTR bob.org.
5 IN PTR bob5.bob.org.
6 IN PTR bob6.bob.org.

[Skipped a few lines]

253 IN PTR bob253.bob.org.
254 IN PTR bob254.bob.org.
255 IN PTR broadcast.bob.org.
```

Whenever you change any of these two files you will need to also update the serial number. Usually the serial number is just another form of the date so

that you always create later numbers. In this example the serial number is the date in the form YYYYMMDD## where ## stands for the version number for the day. It is quite possible to change the serial number for the DNS more than 10 times a day so it is best to have 2 digits to use. Also, whenever you make a change you will need to restart the DNS in order to have the changes take affect.

```
bash# /etc/rc.d/init.d/named restart
```

15.3 Zone Transfers

Zone transfers happen automatically once everything is configured correctly. During a zone transfer the files from one DNS are copied to another DNS and modified slightly. Usually they are sorted. In order for this to happen the secondary server must know the IP address of the master server. Once the IP address is known, the secondary server requests to files. The master server must be set to allow zone transfers to the secondary server. The secondary server compares the serial numbers and gets the master server files only if the serial number is greater the the serial number that is currently on the secondary server. Since the servers operate as user named the files in the **/var/named/** directory and the directory itself need to be owned by named. The secondary server will then use these files until the amount of refresh or expire seconds marked in the file header have passed.

Chapter 16

DHCP Server

The DHCP server gives out IP addresses from a pool of available IP addresses to machines on a network that are set to receive their IP addresses through DHCP. The DHCP server also gives out information about the Subnet Mask, DNS servers, Domain Name, the Gateway, and other information required to work on the network.

16.1 Setting up dhcpd

Lets say that Bob wants to take his network and put the IP addresses from **1.2.3.200** to **1.2.3.250** into a pool to be given out with DHCP. Bob is going to use **bob.org** (**1.2.3.4**) as his DHCP server. There are a few ways this can be done. If you want you can use **linuxconf** and under Control and Configure Linuxconf modules you can activate the **dhcpd** DHCP server configuration. Once this has been activated you can quit **linuxconf** and start it again. Config, Networking, Boot Services you will find the DHCP/BOOTP server. This will allow you to input information that you would like to give out to computers that ask for IP addresses. GUIs are fun, but limited. Currently, this GUI program does not allow things like Static DHCP. Oh, well. I guess we will just have to write the configuration files by hand. Create a file called **/etc/dhcpd.conf** and put something like this inside:

```
server-identifier 1.2.3.4;
default-lease-time 86400;
max-lease-time 172800;
option domain-name "bob.org";
option domain-name-servers 1.2.3.2,
1.2.3.3;
option host-name "bob.org";
option routers 1.2.3.1;
option subnet-mask 255.255.255.0;
subnet 1.2.3.0 netmask 255.255.255.0{
range  1.2.3.200 1.2.3.250;
default-lease-time 86400;
max-lease-time 172800;
option domain-name "bob.org";
option domain-name-servers 1.2.3.2,
1.2.3.3;
```

```
option host-name "bob.org";
option routers 1.2.3.1;
option subnet-mask 255.255.255.0;
}
```

Now that was easy. All we have to do now is start the DHCP server and have it start on reboot:

```
bash# /etc/rc.d/init.d/dhcpd start
bash# chkconfig dhcpd on
```

16.2 Leases

When the DHCP server gives out an IP address it gives it out as a lease for a certain amount of time. This is given in seconds in the default-lease-time variable in the configuration file. The server stores a list of all the leases and their expiration dates in a file called **/var/lib/dhcp/dhcpd.leases**. This is what an entry in the file looks like:

```
lease 1.2.3.221 {
        starts 2 2002/03/26 02:23:59;
        ends 3 2002/03/27 02:23:59;
        hardware ethernet 33:44:55:66:77:88;
        uid 01:23:45:67:89:ab:cd;
        client-hostname "robert";
}
```

Sometime you do not get all of the information listed above, but you always get the starting time, the ending time, and the ethernet address. With the information in the **dhcpd.leases** file you can setup static DHCP where the same IP address is given to the same ethernet address every time.

16.3 Static DHCP

Static DHCP gives a network a little bit more security and stability. When I say a little bit more security I am only referring to security from people who only know how to use DHCP. Those people will not get an IP address so they will not be able to use the network. If they instead just give themselves an IP address this will not do anything. This is what the above DHCP configuration file would look like if you also added two static DHCP entries for **dns** and **dns2**:

```
server-identifier 1.2.3.4;
default-lease-time 86400;
max-lease-time 172800;
option domain-name "bob.org";
option domain-name-servers 1.2.3.2,
1.2.3.3;
option host-name "bob.org";
option routers 1.2.3.1;
option subnet-mask 255.255.255.0;
```

```
subnet 1.2.3.0 netmask 255.255.255.0{
range  1.2.3.200 1.2.3.250;
default-lease-time 86400;
max-lease-time 172800;
option domain-name "bob.org";
option domain-name-servers 1.2.3.2,
1.2.3.3;
option host-name "bob.org";
option routers 1.2.3.1;
option subnet-mask 255.255.255.0;

        host dns      { hardware ethernet 00:11:22:33:44:55; fixed-address 1.2.3.2; }
        host dns2     { hardware ethernet 11:22:33:44:55:66; fixed-address 1.2.3.3; }

}
```

If you only wanted static DHCP for your network you would remove the lines that refer to the DHCP lease range and lease times. These are the three lines you would remove:

```
range  1.2.3.200 1.2.3.250;
default-lease-time 86400;
max-lease-time 172800;
```

It these were gone then the DHCP server would only give out IP addresses to **dns** and **dns2**.

Part III

Appendix

Appendix A

Extra Information

A.1 X Windows

X Windows was written by MIT in the early 1980s. It has since gone through a lot of revisions and is now at version 11 revision 6 (X11R6). From text mode you can start X Windows by typing:

```
bash$ startx
```

This will start you with the default graphical desktop which is probably Gnome. If you would like a little bit more control you can instead type:

```
bash$ xinit
```

This also starts X Windows, but with much less running. To exit X Windows you can press **Ctrl-Alt-Backspace**. If you have your machine starting in Graphical mode this will just log you out and restart the X server. The GUI interface the X provides can help a lot, especially as you first start learning Linux.

A.1.1 Gnome Desktop

Once in the Desktop you will have icons on the left side of the screen, a panel along the bottom, and a background covering the rest. The icons give you quick links to directories, URLs, and devices. You can create your own by pressing the Right mouse button over the background and selecting New and the type of link.

Probably the most important thing to know is how to get a terminal window open. You can move you mouse over the background and press the Right button. Select New then Terminal to get a terminal window open. From the terminal window you have much more control then you get from the GUI. Typing exit in a terminal window will close it.

A.1.2 Backgrounds and Screen Savers

To change your background and screen saver settings move the mouse over the background and press the Right mouse button and select Configure Background Image. This will open up the Gnome Control Center. On the left you will have the different options including Background (selected), Panel, Screensaver, Theme Selector, and Window Manager. You can browse your desktop to find pictures or use the default pictures stored in the /usr/share/pixmaps/redhat directory. The GUI allows you to use colors or to tile, scale, or center an image. If you select Screensaver you will have the random screensaver loaded. This means that it randomly picks screensavers and cycles through them. There are a lot of other screensavers listed. By selecting them you can usually see a demo in the Screen Saver Demo window. Some screensavers do not have demos, but you can see them by Trying them. When you feel adventuresome you can start to modify your Window Manager settings. This changes the whole look of the windows on the screen, so be careful. If you ever completely mess everything up you can delete the .gnome directory directories and the Desktop directory. Then press Ctrl-Alt-Backspace to restart your X server and login again. This is what you would type:

```
bash$ rm .gnome* -R -f
bash$ rm Desktop -R -f
```

This will wipe out all of your Gnome settings and then you can start over again fresh.

A.1.3 Gnome Panel

At the bottom of the screen is the Gnome Panel. By pressing the Gnome foot icon you can get a list of options. This is how you can start lots of applications and games. The other icons in the panel are either applets or launchers that start programs. You can also add icons and applets to your panel by pressing the Right mouse button over the panel and selecting Panel, Add to Panel, then the item you would like to add. There are a lot of applets so I recommend that you try out a lot of them and see what they do.

A.2 Multimedia

The sounds and graphics of multimedia change your experience with computers. Most of the home computers today are much more multimedia dependent than they ever were before. Linux can also provide the multimedia experience.

A.2.1 Setting up Sound and Graphics

The graphical installation should have been done when you installed Linux, but if you want to change some settings you might need to change the setup. Both sound and graphics can be setup fairly easily using the setup command. As root run type:

```
bash# setup
```

If you want to change the graphics settings then you need to select X configuration. For sound select the Sound card configuration. If you do not want to run the setup utility you can run the configuration programs from the shell using these two commands:

```
bash# Xconfigurator
```

Or

```
bash# sndconfig
```

Sometimes the sound configuration does strange things when it probes. If this is the case with your sound card then you might want to use the noprobe option:

```
bash# sndconfig --noprobe
```

Once you have the configuration done you are ready to use the multimedia capabilities of your machine.

A.2.2 Applications

To use graphical stuff there are a lot of applications. Here is a list of some of the graphical applications and short descriptions:

```
ee              Electric Eyes Image Display Program
gqview          GQView Image Display Program
gimp            GNU Image Manipulation Program
xsane           Scanner Program
xpaint          X Painting Application
gtv             Gtv Mpeg Player
```

For sound there are a lot of applications. Here are a few:

```
xmms            XMMS Sound Player
mikmod          Player for .mod, .s3m, xm, etc.
mpg123          Audio MPEG player
gmix            Audio Mixer
xplaycd         CD Player
```

Like other Operating Systems you can find and install other multimedia applications.

Index