Forward

The fourth edition of this text sees quite a few updates and additions. Hopefully it also sees the start of a more collaborative process for the development of the text. In recognition of the fact that one person, with other things to do, simply can't maintain a book such as this I am setting up a number of methods which you can contribute to this book via the 85321 website (http://infocom.cqu.edu.au/85321/).

Those contributions have already started including

- Bruce Jamieson, one of the very first 85321 students, was responsible for most of the content in chapters 4, 8, 10 and 13. Bruce's original work has since been touched up slightly to keep up with the changing face of Linux.
- Chapter 17 of the text is now a copy of the Linux Security HOWTO by Kevin Fenzi and Dave Wreski.
- Janet Jackson has kindly allowed the reuse of one of her articles in the SAGE-AU newsletter in chapter 1.
- There are also a number of documents which have been quoted and/or referenced throughout the text.

While the above are the direct contributions there are many people who have made indirect contributions to the development of this text including the people behind the LDP and other available Linux documentation, the authors of the many books I've read over the years and the people who read this text and provide feedback and suggestions.

Please, if you find an error (no matter how simple), find an explanation not particularly helpful or have a suggestion for an addition to the text please visit the website and participate.

This participation is particularly important given that there are certain to be errors introduced in this edition. The additions were done quickly with a minimum of proof-reading so please report them if you find them.

David Jones

http://cq-pan.cqu.edu.au/david-jones/

Chapter 0
The Overview

Introduction

Welcome to Chapter 0. Yes, I know it is a strange numbering scheme but this chapter is being added early in 2000 and it is quicker to call it Chapter 0 rather than call it Chapter 1 and then have to renumber all the remaining chapters which have been in the text for a few years.

The inclusion of this chapter is due to feedback from previous students in 85321, Systems Administration. It is an attempt to give you an overview of the course and more importantly of computing, Linux and Systems Administration.

Many students have commented that they have felt lost in the detail of Linux without any idea of how it all fits together. Hopefully this chapter will go some way towards solving this problem. Hopefully it will provide some sort of small map and compass so you have some idea of where you are and where you are going.

I am always keen and willing to hear feedback about this text. If you have useful suggestions please feel free to make them via the various mechanisms which are available on the 85321 website, http://infocom.cqu.edu.au/85321/

This chapter will discuss the following

- the course
  A brief overview of 85321 and why it is the way it is. This will also include an introduction to the material we will cover this term.
- the course material
  A really quick explanation of how the 85321 CD-ROM, Website, textbook and other material all fits together (at least how I hope it will).
- solving your problems
  There is one thing both you and I can be sure of this term. At some stage you will have problems with Linux or 85321. This section will provide some hints and tips on how you should go about solving these problems.
- Computers
  Those of you who have not read widely or have experience within the computing industry will think that computing starts and stops with single, stand-alone Windows computers. This couldn't be further from the truth. This section attempts to give you some idea of at least one version of what is out there.
- Linux
  Last but not least the chapter provides a quick overview of Linux, how it works and some of the important concepts you will learn about during this course.

Other Resources

All of the chapters in this text will have a section called Other Resources near the start of the chapter. The idea is to point you to other resources which discuss related material. The 85321 website will maintain a more up to date list of resources which will include comments from people about those resources and a space where you can contribute comments and provide pointers to resources you found useful.

One of the most common references will be to the Linux Documentation Project (the LDP). The LDP is a collaborative project by many people throughout the Linux community to develop high quality documentation about the Linux system. A mirror of the LDP website is included on the 85321 website/CD-ROM.

Other resources which discuss similar material to this chapter includes

- Online lectures 1, 2 and 3 on the 85321 Website discuss some of the same information covered here. Though some of the information may be a touch old.
- HOWTOs
  These are “smallish” which provide guidance on a particular topic. One HOWTO which covers similar material to this chapter is the UNIX and Internet Fundamentals HOW-TO
- Guides
  The LDP also includes a number of guides that are essentially full-blown books (or very close to it). The Linux and Installation and Getting Started Guide contains some good overview material. The Overview of a Linux System from the Linux Systems Administration Guide is also useful. As is the Linux Overview section from the Linux Administration Made Easy Guide (LAME).

The Course

You can get some idea of what 85321, Systems Administration, and to some extent a career as a Systems Administrator from the following poem written by a past 85321 student.

Lament of a Linux Student

Here I sit broken hearted

Loaded X-Windows and

can't get it started

Off I go in a Tizzy

Looks as though tomorrow I'm busy

I can guarantee that most students will at some stage be frustrated, annoyed, depressed and entirely sick of 85321, Linux and anyone responsible for it. This can also be said for a career in Systems Administration.

Many of you may have heard of 85321 from other students and others may well have drawn some conclusions based on your previous experiences in courses I have taught. Hopefully many of them (the bad ones hopefully) won't apply. The experience of past students in this course can be summarised as follows, 85321 is

- enjoyable
- very practical
- a lot of work

Hopefully this year you will find the emphasis more on the first two rather than the last one. There have been a lot of changes made to 85321 for the year 2000. Most have been implemented to reduce the amount of work, increase the enjoyment and make sure that someone who passes 85321 actually knows something about Systems Administration and Linux.

The rationale

Why is the course the way it is? There are lots of reasons which contribute but the main ones are

- You need to learn about Systems Administration.
  Systems Administration is an essential task, especially given the increasing importance of computers. Systems Administration is difficult. Software and untrained people can't be Systems Administrators. Knowing about Systems Administration will make you a better programmer and computing professional, even if you don't find employment as a Systems Administrator.
- People only learn by doing.
  Sure you might be able to recite back to me a whole bunch of facts, commands and concepts and probably even pass an exam. But you won't know how to be a Systems Administrator. To do this you have to experience it.

The last point cannot be emphasised enough. You will learn nothing from this book and course by simply reading about it. You have to get in and get your hands dirty playing around.

What you will learn

The aim of 85321 is to introduce you to the task of Systems Administration, looking after and maintaining complex computer systems. In particular, 85321 aims to produce students who meet the requirements of a Junior Systems Administrator as outlined in the SAGE Job Description booklet (without the 1 or 2 years experience). You can find an excerpt from the Job Description booklet on the 85321 website (http://infocom.cqu.edu.au/85321/).

Figure 0.1. provides a graphical representation of the topics introduced in 85321.

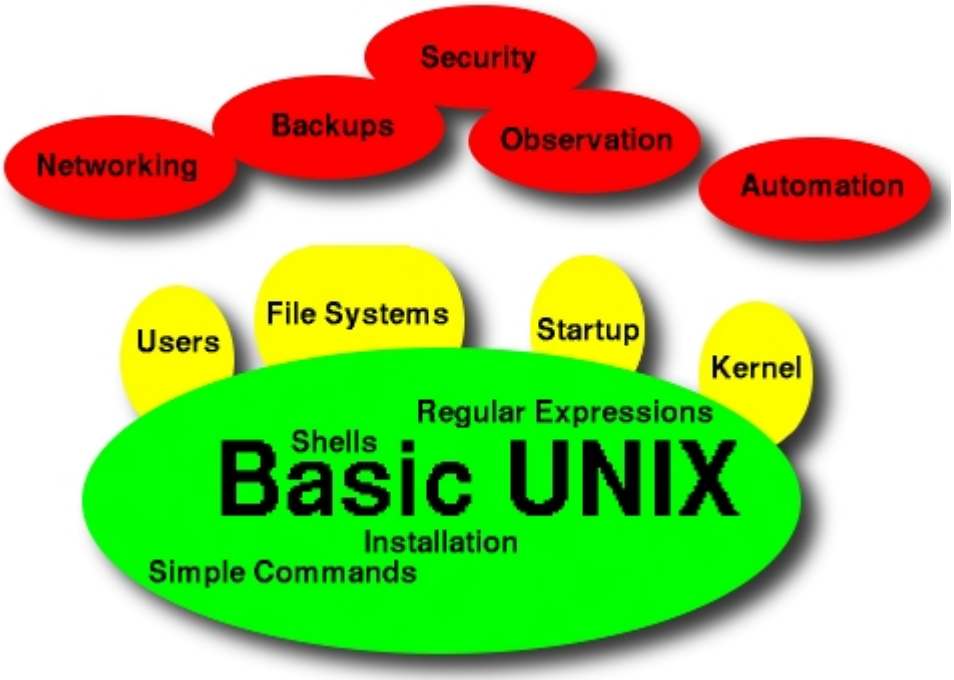


Figure 0.1.
An Overview of the Content of this Book.

For the first eight chapters of this book we concentrate on the foundations, basic UNIX. You need to become familiar with basic concepts such as UNIX commands, shells, and regular expressions before you can progress to the "real" Systems Administration topics. All of these foundation concepts will be reused later in the book.

The next remaining concepts are covered in chapters 9 through 17

- Users and account management
  People have to be able to use the systems you manage. Chapter 9 examines the issues involved with this on a Linux system.
- File systems and Backups
  People use a computer in order to store and manipulate data. That data has to be stored somewhere. Chapters 10 and 11 examine how Linux stores data on hard-drives and how you can perform backups to tape.
- Startup and Shutdown
  Operating systems such as Linux and Windows NT are not simple systems. The process to start them up and shut them down is quite complex and problems can arise. Chapter 12 examines the Linux startup and shutdown process.
- Kernel
  Many of the services provided by a computer are implemented in the kernel of the operating system. Chapter 13 examines how to configure, compile and install the Linux kernel.
- Automation and Observation
  Once your computer is up and running you need to be able to automate tasks and observe what is going on. Chapter 14 examines how to achieve these two tasks on a Linux computer.

- Network
  Without a network connection and network services most modern computers are considered useless. Chapters 15 and 16 examine how to connect, configure and use a Linux computer on a network.
- Security
  Ensuring that your computer and its contents is safe from prying eyes is an essential part of any Systems Administrators job. Chapter 17 provides an overview of security on a Linux system.

All these concepts are essential to Systems Administrators regardless of the type of operating system they are using. Someone managing a Windows NT system still needs to ensure the security of the system, connect it to a network, configure it for new drivers and keep an eye out on what is happening.

Why not NT?

A very common question from 85321 students is why are we using Linux? Why aren't we using NT. Here are some of my answers to those questions.

- It is not cheap
  It costs money to distribute NT to the couple of hundred students doing 85321 in three or four countries. A lot more money than it does to distribute Linux.
- it is not complete
  Adding to the cost is that when you get NT you don't get a real Web server, database and a bunch of other important software.
- it hides its complexity
  Windows NT is supposed to be easy to administer. After all it is all GUI based. That isn't an argument there are similar GUI based tools for managing UNIX boxes. The problem with GUIs, especially when you are learning about systems, is that GUIs hide how things work. As a Systems Administrator you need to know how things work. You don't need to know that to get it to work you press that button on that dialog box. A trained monkey can work that out.
- it is closed
  NT is Microsoft's. They own it. They make the rules. If they are unhappy, the change NT. Linux is owned by a community of people who work together to make it better.
- if you learn Linux you can learn NT
  Lastly, if you can learn the material in this textbook. Learning how to administer an NT computer is no great difficulty.

Course Material

For 85321 you will (should) have access to this textbook, an 85321 CD-ROM and the 85321 Website. This section gives a brief overview of the relationships between this material.

Textbook

This book provides most of the reading and exercises you will need for 85321. You should end up reading most of it. Electronic copies of the text are available on the 85321 website. You can purchase copies of it from the CQU bookshop.

There are a couple of older chapters from this text which are not included with the print version. They are available from the 85321 website/CD-ROM.

85321 CD-ROM

The 85321 CD-ROM will contain a mirror of the 85321 Website. The main use of this CD-ROM is to provide quick access to the website material without requiring you to connect to the Internet.

There most useful parts of the 85321 CD-ROM are expected to be

- the online lectures
  Recorded early in 1999 these online lectures cover the first 5 or 6 weeks of the semester and include audio.
- the Linux Documentation Project mirror
  The LDP is the main source for Linux information. Copies of most of the LDP information is available on the CD-ROM.

The 85321 CD-ROM does not contain a copy of Linux. To install Linux you will need a separate CD.

The 85321 CD-ROM is available for purchase from the CQU bookshop.

85321 Website

It is intended that the 85321 website will be the primary site for interaction and information exchange. The 85321 website should always have the most up to date copies of information.

The 85321 website will also have a number of features which will enable you to make contributions to improving the site and the unit. Please take the time to visit and become familiar with the website and its features. The URL is

http://infocom.cqu.edu.au/85321/

Solving Problems

Students enrolled in 85321 will be nearing the end their degree. It won't be to long until you are computer professionals being employed to do work with computers. When you are a computer professional you will not be able to ask the lecturer how to do something. You will need to know how to solve the problem yourself, to work it out.

If there is one thing I hope you learn from 85321 it is the ability to solve your own problems.

Chapter 2 of this textbook offers more details about how you should go about solving problems. Please refer to it.

Computers in the Real World

Chances are most of your experience with computers are with Wintel PCs (computers with Intel CPUs running various versions of the Windows operating system). As with most people your past experience colours your beliefs. Out in the "real world" (a term I will use throughout the book to refer to largish organisations) there is a lot more to computers than Wintel computers with a single monitor, CPU and keyboard.

It is hoped that this section will introduce you to some of the differences you can expect to experience when you enter the "real world".

What you think computers are

Chances are you think computers have one monitor, one CPU, some RAM, a keyboard, a printer, a couple of other perhiperals and maybe a network connection. To use the computer you sit down in front of it,

- Turn it on.
  As a result the computer finds some boot information on one of the drives. Loads the kernel of the operating system you use, configures the machine and starts up some other software services.
- Get presented with a GUI interface, i.e. Windows, on the monitor.
- Do stuff by double clicking on icons and the like.
  As a result the computer loads programs from file and executes them using your computer's CPU and displays the results on the monitor.
- You might be able to connect to a network drive.
  The network drive might contain data or maybe some programs which you can run using the CPU of your computer.
- When you are finished you turn the computer off.

Some alternatives

This isn't the way it is always. There are computers which differ on any one of the above assumptions. The following introduces you to some of them.

No heads

There are situations where computers can have no monitors at all (often referred to as headless computers). For example the servers produced by Cobalt Networks (http://www.cobalt.com/) which provide file, print and web serving capabilities are essentially blue boxes with a power and network connection. These servers don't have any peripherals connected to them at all. All management is done via the network. Most multimedia developers will work with two monitors. One which has all their "code" and development work, another which shows the end product.

No Graphical User Interface (GUI)

GUIs (graphical user interfaces) such as Windows are resource hogs. Running a GUI takes more RAM and larger CPUs than running a text-based command line. With Windows NT you have to run the GUI. With UNIX you can chose to run a GUI or a command line. Since UNIX isn't burdened by a GUI a much smaller machine can run Linux and do the same job as a larger WinNT machine.

More CPUs

Most personal computers have a single CPU. It is not fairly common for largish network servers to have at least 2 or maybe 4 CPUs. The SUN HPC 10000 (http://www.sun.com/servers/hpc/products/hpc10000.html) will support up to 64 CPUs, 64Gb of RAM and over 64 Terabytes of disk space. Clustering technology, such as Beowulf (http://www.beowulf.org/), allows you to connect multiple personal computers together as a network and treat them as a single computer.

No disks

Managing a large network of computers where users can modify information on the hard-drive can be a lot of work. People make changes, people make mistakes, Systems Administrator must fix mistake. One solution to this is not to allow people to make changes. In some cases the machines don't even have disks. All the information and programs there computer uses comes from the disks of another computer. In the early 90s some Postscript printers actually had more computing power than the personal computers which were sending them print jobs.

Loading programs from a disk on another computer and running them on your own computer is common to both diskless workstations and also to most Windows users. It is common in companies to use a large disk connected to a server for central applications. You want to run Word, well you connect to the network drive that contains Word and run it. The CPU in your machine does the work executing Word but loads it from a network disk.

Sharing CPUs

UNIX is a true multi-user operating system in the sense that someone on another computer can log onto my computer and run programs. In this situation it is the CPU of my computer (hopefully a large server) which runs the program while the new person's computer takes care of handling the input and output. Under UNIX this can be achieved using telnet for text based programs (this feature is available under Windows) and using features of the X Windows system for GUI-based programs.

But alas all is not lost. Virtual Network Computing (VNC) http://www.uk.research.att.com/vnc/ is freely available and provides a similar capability for Windows and Mac computers. In fact, it allows any UNIX, Windows, Mac computer to run applications on any UNIX, Windows Mac

computer and have the output appear on the original computer. It even allows the same effect to be achieved via any Web browser which supports Java.

Multiple operating systems, one at a time

Up until this course most of you will have been using a single operating system on your computer. It is possible to have more than one operating system on the one computer. The standard approach to achieving this is placing each operating system on its own partition and when you first turn the computer on choosing which operating system you want to run, e.g. WinNT, Linux or Win98.

With the standard configuration you cannot have more than one operating system running at any one time. Mainly due to the fact that the operating system provides the interface between the hardware and user programs. So each operating system takes over the hardware. Operating systems have not been able to watch Sesame Street to learn how to share.

Running programs from one operating system on another

Usually you cannot run a Windows NT program on a computer running Linux computer or a Linux program on a computer running Windows NT. However, there are are "systems" which aim to allow you to achieve this. The most common under Linux is the Wine system (http://www.winehq.com/) which allows you to run Windows binaries under Linux.

Multiple operating systems at the same time

In some instances you have to have access to more than one operating system. The above three solutions are workable but have their drawbacks. Using a system like VNC means that you need to have more than one computer. Running multiple operating systems, one at a time, means you have to reboot your computer to change operating systems. The WINE approach isn't quite ready for prime-time use.

An alternative approach is provided by VMware (http://www.vmware.com/). VMware provides software which supplies a virtual machine on which you can run other operating systems within other operating systems. For example, using VMware for Linux you can run a copy of Windows NT on the VMware virtual machine and then run any Windows application at the same time as running Linux and Linux applications.

An Overview of Linux

The research literature into education is full of discussion about the advantages of mental models provide towards making learning easier. The idea is that you can only really learn or know something if you build a correct model of the concept you are learning about in your mind. In other words, you will find 85321, Systems Administration and Linux much easier if you have a good idea of how Linux and its various components all work. Hopefully the following will aid you in achieving this goal.

To achieve this goal the section will be divided into four sections

- booting
  Which describes what happens when a Linux machine boots. "Boots" is just computer jargon for starting up. Chapter 12 of the text explores the boot process in more detail.
- running
  Once a Linux computer has booted it enters another phase, the running phase. This section looks at the various parts of the Linux operating system and supporting tools which enable a Linux computer to "run".
- Shutting down
  This section completes the circle. At some stage you will want to turn the Linux computer off. This section describes what happens when you shut it down.
- The layers
  Much of the content and concepts introduced in this text will fall into one of a number of layers. This section outlines those layers and offers a brief explanation.

Booting

When you turn a Linux computer on the following happens (similar steps happen for Windows NT and other types of computer)

- it executes instructions contained within some read only memory (ROM)
- this usually results in the computer looking in a few places on disks for a boot sector
- the computer loads the boot sector and executes the instructions contained in that boot sector
- these instructions generally load the Linux kernel
- the Linux kernel checks the available hardware, attempts to configure it and then starts up two processes, swap and init
- the `init` process then starts executing a bunch of shell scripts contained in the `/etc/rc.d` directory
- the startup scripts perform various configuration steps and start a number of daemons

At this stage your Linux computer is running. It is usually in one of a number of run levels.

At any one of these stages problems can occur which cause problems. For example, a corrupt boot sector will mean it can't be executed, a kernel which is incompatible with the installed hardware will not be able to use that hardware. As a Systems Administrator you will be responsible for diagnosing and fixing these problems.

Running

Once the computer is up and running you need to start using it. You will need some type of interface to issue commands to the computer and see the results. At the most general level Linux has to types of interface

- text interface
  A simple command-line interface is provided by a variety of shells which allow you to enter commands via the keyboard. This is the original interface to UNIX.
- A graphical interface
  Linux comes with X Windows which provides all the functionality and features required to implement a graphical user interface (GUI).

The GUI provided by X Windows is significantly more flexible than that of Windows or the Mac.

The commands or programs you execute are all stored on disk in files. Rather than stick all those files in together they are separated out using directories. File and directory mean much the same sort of thing as document and folder. Chapters 4, 5 and 10 provide more information about how information is stored in files and directories and the commands you can use to manipulate them.

There are a large number of functions which are common to a lot of programs. For example, opening a Window in a GUI, print some text and opening a file. Rather than have every program write their own code to do this Linux comes with a large collection of libraries. These libraries are stored in common locations (e.g. the directories lib /usr/lib and others) and are referred to when needed.

The operating system provides a number of low level tasks such as memory management, CPU management, device drivers and the like. Programming libraries provide the executable code to perform slightly higher level tasks required by other programs, such as printing to the screen. Services such as logging onto the system, handling network connections or running the startup scripts are performed by daemons.

A daemon is simply a program. It gets started up, usually by the startup scripts when the computer starts, and then sits around waiting for some interesting event to occur. When that event occurs it examines the event performs some appropriate task and then goes back to sleep.

There are a large number of daemons on a UNIX system and a fair amount of Systems Administration is dealing with the management and configuration of daemons. This could be quite complex. Thankfully all daemons behave much the same. They generally all have

- a configuration file
  Under UNIX most configuration files are text-based. This is good because text is easy to edit and manipulate with normal text processing tools. The configuration file essentially tells the daemon what to do, when and sometimes how. The first thing a daemon will do is read its configuration

file. If you change the configuration file you have to tell the daemon to take another look at the configuration file either by stopping and restarting it or sending it a signal (more on these later)

- An executable program
  This is the daemon. The thing which is started and then listens for interesting events.
- A log file
  As the daemon performs tasks it will normally record what it is doing in a log file. This allows the Systems Administrator to find out what happened. It is also the primary tool for figuring out what went wrong.

Processes, programs in execution, perform all work on a UNIX system. A process is essentially a bunch of operating system data structures, code and data. The data structures for each process keep a track of the identity of the person who ran the process (in the process' user id). The process will only be allowed to perform tasks that the process' owner has been allocated.

There is one person (account), the root account, which can do anything because permissions are not checked for root. The root account is usually only available to the Systems Administrator. The lack of control placed on the root account means that it is not a good idea to be using the root account to perform normal tasks.

Shutting down

You can't simply turn a UNIX, or for that matter any reasonably complex, computer off. You have to let it have time to tidy up and finish various tasks so it can shutdown in a safe state.

The shutdown process involves the `init` daemon, the same daemon involved in the startup process, executing a number of shell scripts to safely close down the services it started during the boot process.

Layers

A computer system, especially one running Linux, can be though of containing a number of different layers including

- Hardware
  At the lost levels is the physical equipment that provides the basic functionality required for the system to perform. At various stages you will be adding or removing hardware from your system.
- Device drivers
  The next step up are the device drivers that form part of the Linux kernel. These devices drivers are essentially programs that know how to communicate with specific devices. If you want to use a particular piece of hardware you must have an appropriate device driver included with the kernel of your system.
- Other kernel services
  The kernel also provides a number of slightly higher-level services that

have little to do with talking directly with hardware. Examples include CPU scheduling, memory management and file systems. If you want your Linux system to be able to read Windows-98 floppy disks then certain services are required to be included with your kernel.

- Device files
  Outside, but closely related to device drivers, are device files. Device files provide a standard interface to devices. This common interface is often used by processes from the next two layers to communicate with the device. Device drivers are little more than translators between the hardware device and the Linux device file interface. No device file often means you can't use the device, even if you have an appropriate device driver.

- Daemons
  As introduced earlier daemons are processes that start and then wait some event to occur. Daemons are often used to implement system level services such as starting up the Linux system and allowing people to log on to your Linux system.

- User programs
  At the top level are the user programs. These user programs make use of the services provided by daemons, device files and other kernel services to perform tasks required by people. Some of these user programs provide the interface people use, e.g. shells and the X-Window system.

What's the use of all these layers? Why should I bother understanding them? Well it makes it much easier to identify and fix the problem. Working your way up the layers (from hardware up to user programs) can often be a good approach to solving problems.

For example, let's assume I have been unable to connect to my Linux computer over the network. How can I identify and solve the problem

- Hardware
  First step is to make sure the hardware is working. For example, is the network connector plugged in, is the Linux computer turned on, can I connect to other similarly located computers.

- Device drivers
  Does the kernel on the Linux system contain the appropriate device drivers for the network hardware I am using.

- Kernel services
  Are the kernel services required to connect to remote computers correctly installed and configured? Are other similar network services working?

- Device files
  Do the appropriate devices exist? Have they been configured correctly?

- Daemons
  Do the log files of the associated daemons show any errors? Is the required daemon executing?

And so on. Hopefully you get the idea of slowly progressing up the layers enables you to rule out possibilities.

Hopefully the remainder of this text will provide you with the information necessary to now which kernel services are associated with which features of Linux.

Conclusions

Computing is a large field with many different tasks implemented with a plethora of approaches. This chapter has provided a small list of some of the possibilities. These aren't the only ones and there are sure to be some new ones developed. As a computing professional you need to be aware of the possibilities.

You will only learn Systems Administration by doing Systems Administration.

Chapter 1

The What, Why and How of Sys Admin

A beginning is the time for taking the most delicate care that the balances are correct.

-- Frank Herbet (Dune)

Introduction

Systems Administration is one of the most complex, fulfilling and misunderstood professions within the computing arena. Everybody who uses the computer depends on the Systems Administrator doing their job correctly and efficiently. However the only time users tend to give the Systems Administrator a second thought is when the computer system is not working. A broken computer system implies some fault on the part of the Systems Administrator..

Very few people, including other computing professionals, understand the complexity and the time-consuming nature of Systems Administration. Even fewer people realise the satisfaction and challenge that Systems Administration presents to the practitioner. It is one of the rare computing professions in which the individual can combine every facet of the computing field into one career (including programming).

The aim of this chapter is to provide you with some background to Systems Administration so that you have some idea of why you are reading this and what you may learn via this text.

What Systems Administrators do

Systems Administration is an old responsibility gaining new found importance and acceptance as a profession. It has come into existence because of the increasing complexity of modern computer systems and networks and because of the economy's increasing reliance on computers. Any decent size business now requires at least one person to keep the computers running happily. If the computers don't work the business suffers. Smaller companies will don't have the size to support a full-time Systems Administrator and will likely share one (usually some form of consultant) amongst a number of other companies.

It can be said that Systems Administrators have two basic reasons for being

- ensuring that the computing system runs correctly and as efficiently as possible, and
- ensuring that all users can and do use the computing system to carry out their required work in the easiest and most efficient manner.

People who have studied operating systems may remember these two reasons as being similar to the responsibilities of operating systems. You may also

remember from operating systems that these two responsibilities often conflict with one another.

Users will want things a specific way which may not be the best for the organisation. For example, Joe Bloggs in accounting may want this program installed, however the organisation may already have a site licence for another program. The System Administrator, with help from policies, documentation a number of other resources, must attempt to balance these two conflicting aims.

Why do we need them

Every year some company, over the last couple of years it is usually Microsoft, announces some new product which is going to make Systems Administrators obsolete. In fact a couple of the network devices mentioned in Chapter 0 rarely need any form of Systems Administration activities, you set them up and they run.

The reason for this is that these types of devices are designed to do one job, e.g. Mail/file/print servers, and nothing else. Their purpose is very specific. However, most organisations cannot be that specific about what they want their computers to do and chances are there won't be a computing device that does exactly what the organisation wants.

A lot of the need for Systems Administration is to bridge the gap between what people/organisations want to do and what the computers the organisation has can do.

What they do

The real work required to fulfill these aims depends on the characteristics of the particular computing system and the company it belongs to. Factors that affect what a Systems Administrator needs to do come from a number of categories including: users, hardware , support and policy.

Users

Users, your colleagues and workmates that use computers and networks to perform their tasks contribute directly to the difficulty (or ease) of your task as a Systems Administrator. Some of the characteristics of people that can contribute to your job include:

- How many users are there?
  Two hundred users are more difficult to help than two users and also require completely different practices. With two, or even ten/twenty, users it is possible to become well known to them and really get to know their requirements. With two hundred, or in some cases two thousand users, this is simply not possible.
- The level of the user's expertise.
  This is a combination of the user's actual expertise and their perceived expertise. A user who thinks they know a lot (but doesn't really) can often be more trouble than a user who knows nothing and admits it.

Users who know what they know.

Picture it. You are a Systems Administrator at a United States Air Force base. The people using your machines include people who fly million dollar weapons of destruction that have the ability to reduce buildings if not towns to dust. Your users are supremely confident in their ability.

What do you do when an arrogant, abusive Colonel contacts you saying he cannot use his computer? What do you say when you solve the problem by telling him he did not have it plugged in? What do you do when you have to do this more than once?

It has happened.

- What are the users trying to do?
  If the users are scientists doing research on ground breaking network technology you will be performing completely different tasks than if your users are all doing word processing and spreadsheets.
- Are they responsible or irresponsible?
  Do the users follow the rules or do they make their own? Do the users like to play with the machines? Being the Systems Administrator in a computing department at a University, where the users are computing students who want to play and see how far they can go is completely different from working in a government department, where the users hate computing and only use them when necessary.
- Who do the users know?
  A user, who has a 15-year-old, computer nerd son can often be the cause of problems since the son will tell the parent all sorts of things about computers and what can be done. Very few people have an appreciation of the constraints placed on a Systems Administrator and the computers under their control. Looking after a home PC is completely different to managing a collection of computers at a place of work.

Hardware/Software

The computers, software, networks, printers and other peripherals that are at a site also contribute to the type and amount of work a Systems Administrator must perform. Some considerations include:

- How many, how big and how complex?
  Once again greater numbers imply more work. Also it may be more work looking after a network of Windows NT machines than a collection of Windows 3.1 computers. Some sites will have supercomputers, which require specialised knowledge.
- Is there a network?
  The existence of a network connecting the machines together raises additional problems and further increases the workload of the Systems Administrator.

- Are the computers heterogeneous or homogenous?
  Is the hardware and software on every machine the same, or is it different. A great variety in hardware and software will make it much more difficult to manage, especially when there are large numbers. The ability to specify a standard for all computers, in both hardware and software, makes the support job orders of magnitude easier.

Support

One other area, which makes a difference to the difficulty of a job as a Systems Administrator, is the level of support in the form of other people, time and resources. The support you do (or don't) receive can take many forms including:

- Are you alone?
  At some sites there is one administrator who does everything from installing peripherals, fixing computers, doing backups, helping users find the enter key and a range of other tasks. At other sites these tasks are split amongst a range of administrators, operators and technicians.
- Are you a full time administrator?
  In some cases the administrator looks after the machines in addition to performing their "real job".

What are the feelings of staff and management towards the Systems Administrators?
In many companies the management and staff see Systems Administrators or other computer support people as overhead. This impression of Systems Administrators as an unnecessary expense influences how the users will act. Similar feelings can occur if previous Systems Administrators have been unprofessional or unable to perform their jobs.

Policy (Management)

As you read through this text you will be introduced to various forms of policy about the use of computers and networks which are needed. These policies define the what, why and how things are done within an organisation. These can be as petty as always using a specific template for letters, memos and faxes through to something as important as whether or not management can order the Systems Administrator to read another employee's email..

Official policies are usually the responsibility of management. It is they who should define the rules and the Systems Administrator who puts them into action. Obviously policy shouldn't be made in a complete vacuum without any knowledge of what is possible (but it often is). Additionally these policies should exist and the people using the systems should be aware of them. If this isn't the case you, or the organisation, can be in trouble legally if you wish to enforce a rule (e.g. You can't send pornographic material to the staff mailing list).

Home and the Real World

Chances are that your only experience with computing is what you have gained maintaining your computer at home or perhaps helping out a few friends. While useful this experience does not prepare you for what computing is like in the real world, especially in a largish organisation. This small section, along with repreated attempts throughout the remaining chapters of this book (see the Computers and the Real World section in Chapter 0), attempts to provide you with some idea of what is involved with computing in the "real world".

Some of the differences you will face in the real world include

- numbers of users
  Most Systems Administrators will be responsible for looking after organisations with somewhere between 10 up to 1000s of users. Looking after a small number of users who you know is simple. You can let each person do their own thing and the workload won't be too great. However, with 100s of users you have to standards and policies which are kept. Otherwise you will spend all your time trying to remember the differences and be unable to do some real work.
- 24x7 operation
  Increasingly organisations are finding that they must have computer systems available 24 hours a day, 7 days a week. Maintaining this sort of availability requires a number of special steps and rules out a lot of practices which are okay when 24x7 operation isn't an issue. As you progress through the text think about what is written. What implications would 24x7 operation have on the concepts you are reading about.

What Sys Admins need to know

The short and sweet answer is that to be a really good Systems Administrator you need to know everything about the entire computer system including the operating system, hardware, software, users, management, network and anything else you can think of that might affect the system in any way.

Failing that lofty aim the System Administrator must have the ability to gain this all-encompassing knowledge. The discovery process may include research, trial and error, or begging. The abilities to learn and problem solve may well be the two most important for a Systems Administrator.

At some time during their career a Systems Administrator will make use of knowledge from the following (far from exhaustive) list of fields, both computing and non-computing:

- programming,
  Systems Administrators have to be able to program. They might have to write scripts that automate regular tasks or a Visual Basic program to help users perform certain tasks.

- hardware maintenance and installation,
  This may installing new hardware , cleaning old hardware so that it continues to work or diagnosing problems with hardware.
- documentation,
  An essential part of Systems Administrations. Not only must you write documentation for the users of your systems so that they know how to do things. You must also write documentation about what it is you are doing and how you are doing it. This documentation will be used by you and your fellow Systems Administrators.
- testing,
  Testing is not an ad hoc process where you try a few things. It is an indepth field on its own. Systems Administrators have to have some idea about testing. You can't put together a system for 1000 users without performing some sort of testing.
- Human Computer Interface,
  Writing GUI or Web-based applications are a common task for Sys Admins. Both require some sort of idea about HCI issues.
- networks and computer communication,
  Networks are an essential part of any computer system these days. You must be aware of the network and data communications.
- user education,
  When Office 2000 comes out do you think all the workers in an organisation teach themselves how to use it? Chances are the Systems Administrator will have to perform some form of training. If you are lucky your organisation might have professionals who look after this form of training. If you are really lucky your organisation might recognise the importance of paying for this training. I wouldn't hold my breath.
- diplomacy,
  What happens when the second in charge of an organisation tells you that you're an &%^#$# idiot and shouldn't be working here? Scream back, resort to violence, or run away? A Systems Administrator must be a good talker and able to deal with stressful situations.
- legal issues and contracts,
  Unlike many University students most organisations pay for their software (and hardware). This usually involves dealing with some form of licence and legal contracts. Familiarity with these can be very helpful.
- detective work and problem solving,
  Following the virtual crumbs to find the cause of a problem can be a lot like detective work.
- management and policy setting, and
- public relations.

Reading

The Systems Administrators Guild (SAGE, http://www.usenix.org/sage/) is a professional association for Systems Administrators. SAGE has developed a job description booklet that helps describe what Systems Administrators do and what they need to know.

A summary of this book is available from the 85321 Web site/CD-ROM under the Resource Materials section for week 1.

This text and the unit 85321 aim to develop Junior Systems Administrators as specified in the SAGE job descriptions booklet, without the 1 to 3 years experience.

Why UNIX?

Some parts of Systems Administration are independent of the type of computer being used, for example handling user complaints and getting on with management. However by necessity there is a great deal of complex platform dependent knowledge that a Systems Administrator must have in order to carry out their job. One train of thought is that it is impossible to gain a full understanding of Systems Administration without having to grapple with the intricacies of a complex computer system. This is something I believe.

This text has been written with the Linux operating system (RedHat version 6.1), a version of UNIX that runs on IBM PC clones, in mind. To get the most out of this book you will need access to the root password of a computer running RedHat version 6.1.

The reasons for choosing UNIX, and especially Linux, over any of the other available operating systems include

- UNIX has a long history both in industry and academia.
- Knowing UNIX is more likely to help your job prospects than hinder them.
- UNIX/Linux is one of the current industry buzzwords.
- It is hardware independent.
- Linux is free
  A CD with RedHat Linux can be purchased from the CQU bookshop for less than $(AUD)10.
- Linux runs on a cheap, popular type of computer.
  A 386 with 16Mb of RAM can provide mail, web, print and file services for up to 25 users. 486 with 32Mb for up to 100 users.
- Linux provides the operating system and almost all the other software you require to set up a computer system for a small organisation.

With Windows NT you will have spend a few thousand dollars, on top of what you spend for the operating system, for a database, web server and other necessary software.

- If you can learn Linux then learning Windows NT is a piece of cake (and uses many of the same ideas).

Just as there are advantages in using UNIX there are also disadvantages. "My Operating System is better than yours" is a religious war that I don't want to discuss here.

UNIX past, present and future

The history of UNIX is an oft-told tale and it is sometimes hard to pick the right version. The story has been told many ways and the following is one version. Being aware of the history can provide you with some insight into why certain things have been done the way they have

Unix History

These readings are on the 85321 Web site (or CD-ROM) under the Resource Materials section for week 1.

At the current point in time it appears that UNIX has ensconced itself into the following market niches

- server operating system, and
  Machines running UNIX are acting as file servers and network servers for local area networks (LANs) of smaller client machines (running MS-DOS, Windows, or Macs).
- workstation operating system.
  Workstations are nominally powerful computers usually used by a single user. Engineers, scientists and other people who require a lot of computing power generally use them.

Both these roles are being challenged by the arrival of new operating systems like Windows NT.

Linux is slowly making inroads into the personal computing environment. I know of a few companies who now use PCs running Linux, X-Windows and Gnome/KDE as the standard desktop.

Linux

This book has been specifically written to centre on the Linux operating system. Linux was chosen because it is a free, complete version of the UNIX operating system that will run on cheap, entry level machines. The following reading provides you with some background into the development of Linux.

Linux: What is it and a history

These readings are available on the 85321 Web site (or CD-ROM) under the Resource Materials section for week 1.

The relationship between Linux and UNIX

Linux is by no means the only version of UNIX currently available. For example you will find at least three other versions of UNIX being used at CQU

- Solaris
  The product of Sun (http://www.sun.com/). These are the guys also responsible for Java. Sun are probably the major commercial UNIX vendor today.
- Tru64 UNIX
  The product of Compaq Computer who bought out the original manufacturers Digital.
- HP/UX
  Produced by Hewlett Packard

While almost all of the specifics covered in this text and the course 85321 are Linux specific the general concepts are applicable to most versions of UNIX. For example the magic file for Linux may be located in /usr/share/magic but on Solaris it may well be in a different location. However, the concept of a magic file still exists on Solaris.

The trick is to remember you will become experienced with Linux specific information. While another version of UNIX will be different you should be able to pick it up quite fast.

Some more Sys Admin theory

Systems Administration is not a responsibility specific to the UNIX operating system. Any company that relies on computers must have Systems Administrators. They may not call them Systems Administrators but studies have shown that it is cheaper to have a full time professional maintaining a company's computers than it is to expect the computer users perform the same tasks.

Many of the tasks of Systems Administration are not platform specific. For example a recent survey of Systems Administrators found that 37% of an administrator's time is spent helping users. This chapter examines some of the important platform independent tasks that a Systems Administrator must perform. Any Sys Admin that ignores these tasks is going to be in trouble very quickly.

For the purposes of this chapter these tasks have been divided up into four categories

- daily operations,
- hardware and software,
- interacting with people, and
- administration and planning.

Daily operations

There are a number of tasks that must be done each day. Some of these tasks are in response to unexpected events, a new user or a system crash, while others are just standard tasks that must be performed regularly.

Automate, automate and automate

A priority for a Systems Administrator must be to automate any task that will be performed regularly. Initially automation may take some additional time, effort and resources but in the long run it will pay off. The benefits of automation include

- no need to reinvent the wheel,
  Everytime you need to perform the task you don't have to remember how to do it.
- it is much simpler,
- it can be delegated,
  If the task is simple it can be delegated to someone with less responsibility or it can be completely automated by using the scheduling capabilities of `cron` (introduced in a later chapter).

For example

Obvious examples for automation include

- adding and removing users,
- performing backups, and
- checking disk usage.

System monitoring

This responsibility entails keeping an eye on the state of the computers, software and network to ensure everything is working efficiently. Characteristics of the computer and the operating system that you might keep an eye include

- resource usage,
- what people are doing,
- whether or not the machines normal operations are working.

Resource usage

The operating system and the computer have a number of different resources including disk space, the CPU, RAM, printers and a network. One indication of problems is if anyone person or process is hogging one of these resources. Resource hogging might be an indication of an attack.

Steps that might be taken include

- killing the process that is hogging the resource,
- changing the process' priorities,
- getting more of the required resource.

What are people doing?

As the Systems Administrator you should be aware of what is normal for your site. If the managing director only ever connects between 9 to 5 and his account is currently logged in at 1 in the morning then chances are there is something wrong.

Its important not only to observe when but what the users are doing. If the secretary is all of a sudden using the C compiler then there's a good chance that it might not be the secretary.

Normal operations

Inevitably there will be problems with your system. A disk controller might die, a user might start a run away process that uses all the CPU time, and a mail bounce might result in the hard-drive filling up or any one of millions of other problems.

Some of these problems will adversely effect your users. Users will respect you more if they don't have to tell you about problems. Therefore it is important that you maintain a watch on the more important services offered by your computers.

You should be watching the services that the users use. Statistics about network, CPU and disk usage are no good when the problem is that the users can't send email because of a problem in the mail configuration. You need to make sure that the users can do what they normally do.

Hardware and software

Major tasks that must be performed with both hardware and software include

- evaluation,
  Examining different packages and deciding which is the best for your company's purpose.
- purchase,
  Actually obtaining the software, spending the money and signing the contracts.

- installation,
  Placing the hardware or software onto your system and making it available to the appropriate users.
- testing and maintenance,
  Making sure the equipment works and keeping it working.
- upgrading,
  Modifying the product to a later version.
- phasing out.
  Removing the product from use at your company.

At many companies the Systems Administrator may not have significant say in the evaluation and purchase of a piece of hardware or software. This causes problems because hardware or software is purchased without any consideration of how it will work with existing hardware and software.

Evaluation

It's very hard to convince a software vendor to allow you to return a software package that you've opened, used but found to be unsuitable. The prospect of you making a copy means that most software includes a clause that once you open a packet you own the software and your money won't be refunded.

However most vendors recognise the need to evaluate software and supply evaluation versions. These evaluation versions either are a stripped down version with some features turned off, or contain time bomb that makes the package useless after a set date.

Purchase

Under UNIX there are basically two types of software

- commercial software, or
- shareware, public domain or free software.

Commercial UNIX software will come with the standard agreements and may also include a user limit. The software might be able to be used by 4 or 5 users simultaneously. Most commercial software is managed by licensing software that controls how many copies are being used. As part of the purchase you will receive license numbers that govern how the software may be used.

It must be remembered that free software is never free. It still requires time to install, maintain and train users. All this can add up. Some free software can be incredibly easy to install and maintain.

Installation

Most sites will have a policy that covers how and where software must be installed. Some platforms also have software that makes the installation procedure much simpler. It is a very good idea to keep local software separate from the operating system distribution. Mixing them up leads to problems in future upgrades.

Under Linux and many other modern Unices it is common practice to install all software added locally under the directory `/usr/local`. There will be more on software installation in a later chapter.

Hardware

At some sites you may have technicians that handle most of the hardware problems. At some sites the Systems Administrator may have to everything from preparing and laying cable through to fixing the fax machine. Either way a Systems Administrator should be capable of performing simple hardware related tasks like installing hard drive and various expansion cards. This isn't the subject to examine hardware related tasks in detail. The following however does provide some simple advice that you should keep in mind.

Static electricity

Whenever you are handling electrical components you must be aware of static electricity. Static can damage electrical parts. Whenever handling such parts you should be grounded. This is usually achieved by using a static strap. You should be grounded not only when you are installing the parts but at anytime you are handling them. Some people eagerly open packages containing these parts without being grounded.

Powering down and wiggling

Many hardware faults can be fixed by turning the system off (powering down) and either pushing on the offending card or SIMM (wiggling). Sometimes connectors get dirty and problems can be fixed by cleaning the contacts with a soft pencil eraser (in good condition).

Prevention

Regular maintenance and prevention tasks can significantly reduce the workload for a Systems Administrator. Some of the common prevention tasks may include

- ensuring that equipment has a clean, stable power supply,
  Using power conditioners or uninterruptable power supplies (UPS) to prevent power spikes damaging equipment.
- ensuring equipment is operating at appropriate temperatures,
  Make sure that the power vents are clean and unblocked and that air can actually circulate through the equipment.
- some equipment will need regular lubrication or cleaning,
- making sure that printers are clean and have sufficient toner, ink etc.

Administration and planning

This is a task that often receives less attention than others. However it is an essential task that can critically effect your performance as a Systems Administrator. One of the most important aims for a Systems Administrator is to be pro-active rather than reactive. It's very hard for your users to respect you if you are forever badly organised and show no planning ability.

Important components of administration and planning include

- documentation,
  Both for yourself, the users and management.
- time management,
  This is an essential ability for a Systems Administrator who must balance a small amount of time between a large number of simultaneous tasks.
- policy,
  There must be policy on just about everything at a site. Having policies that have been accepted by management and hopefully the users is essential.
- self-education,
  Computing is always changing. A Systems Administrator must keep up with the pack.
- planning,
  What are the aims for your site and yourself for the next 12 months? What major events will happen for which you must prepare?
- automation, and
  Anything that can be should be automated. It makes your job easier and gives you more time to do other things.
- financial planning and management.

Documentation

Documentation is the task that most computing people hate the most and yet is one of the most important tasks for a Systems Administrator. In this context documentation is more than just documentation for users showing them how to use the system. It includes

- keeping a log book that records all changes made to the system,
- keeping records and maps of equipment, their location, purchase details etc,
  Where all the cables are in your building. Which cables connect where. Where are all the machines physically located.
- labelling hardware,
  When you are performing maintenance on computers you will need to know information like the type of hard drive controller, number and size of disks, how they are partitioned, hostnames, IP addresses, names of peripherals, any special key strokes or commands for the machine (e.g.

how to reset the computer) and a variety of other information. Having this information actually on the machine can make maintenance much easier.

- producing reports,
  Producing reports of what you are doing and the functioning of the machines is extremely important and will be discussed in more detail later.
- taking minutes at meetings, and
  Chances are you will have to attend meetings. Organising, running and recording the minutes of a meeting are all essential skills.
- producing documentation on how to use the systems at your site.
  The standard type of documentation required by both users and other Systems Administrators.

Why keep records?

It is not unusual for a Systems Administrator to spend two to three days trying to fix some problem that requires minor changes to obscure files hidden away in the dim, dark recesses of the file hierarchy. It is not unusual for a problem of this sort to crop up unexpectedly every six to twelve months.

What happens if the Systems Administrator didn't record the solution? Unless he or she is blessed with a photographic memory there is liable to be another two to three days lost trying to fix the problem.

Records of everything done to the system must be kept and they must be accessible at all times.

What type of records?

It is typical for a Systems Administrator and/or a computer site to maintain some type of logbook. There is no set format to follow in keeping a logbook.

There are two basic types of logbooks that are used.

- electronic, or
  Log information is stored using some type of program or by simply creating a file.
- paper based.
  Some form of book or folder in which entries are written by hand.

Table 1.1. compares these two forms of logbook.

Electronic		**Paper**	
For	**Against**	**For**	**Against**
easy to update and search	if the machine is down there is no access to the log	less prone to machine down time	harder to update and search
easy to include command output	can be hard to include diagrams	can be carried around	can become messy and hard to read

Table 1.1.
Electronic versus paper log books

What to record?

Anything that might be necessary to reconstruct the current state of the computing system should be stored. Examples of necessary information might include

- copies of policy regarding usernames, directory structure etc,
  Your site might have a set way of assigning usernames or particular locations in which certain types of files need to be placed.
- diagrams of the physical connections and layout of the machines and network,
  Any information required to reconstruct your system, for example CMOS settings on an IBM PC.
- a copy of a full listing of the device directory,
  The `/dev` directory is likely to contain information specific to your machine. If this directory is trashed having a copy in your logbook will enable you to reconstruct it.
- copies of major configuration files,
- daily modifications to configuration or other files,
- lists of useful commands, and
- solutions to common problems.

Example Log Book Layout

The type of information recorded will depend on your responsibilities and the capabilities of your site. There might be someone else who looks after the physical layout of the network leaving you to worry about your machine.

It is possible that a logbook might be divided into separate sections. The sections might include

- configuration information,
  Listings of the device directory, maps of network and cabling information, and any other static information about the system
- policy and procedure,
  A section describing the policy and procedures of the particular machine (usernames, directory locations etc).
- useful commands, and
  A list of commands or hints that you've come across that are useful and you would like to remember.
- daily modifications.
  The daily modifications made to the system in the normal course of events. The main reason to store this information is so that you have a record of what is being done to your system.

Each entry in a logbook should contain information about time, date, reason for the change, and who made the change.

If you intend using a paper based logbook then one suggestion is to use a ring binder. Using a ring binder you can add pages to various sections if they start to fill up.

Policy

Think of the computer systems you manage as an environment in which humans live and work. Like any environment, if anarchy is not to reign supreme then there must exist some type of behavioural code that everyone lives by. In a computer system this code is liable to include such things as

- a single person shall not hog all the resources (disk, cpu etc),
- users who work for accounting have xyz access, those who work for research have zyx access, and
- no-one should endeavour to access areas in which they are not allowed.

Penalties

A set of rules by themselves is not enough. There must also exist

- a set of penalties to be applied if one of the policies is broken,
- a person(s) charged with detecting the breaking of policy,
- a person(s) charged with deciding the appropriate policy,
- a mechanism for the change of policy and penalties, and
- a mechanism for informing users of the policy and the penalties.

If any one of these necessary components is missing the system may not work to the best of its ability.

It is essential that every computer site have widely recognised and accepted policies. The existence of policies ensure consistent treatment of all cases. Policies provide guidelines of what to do in particular cases and what to do if the policies are broken.

Types of Policy

The types of policies you might want to have include

- the level of service you provide,
  What operating systems, software etc that you can and will support. What services you provide your users. When will the Systems Administrators or help desk available.
- the rights and responsibilities of the users, and
  What they can and can't do. What happens if they break those rules.
- the rights and responsibilities of the administrators.
  An often over looked policy. Should Systems Administrators look at other people's mail?

Creating policy

Creating policy should include many of the following steps

- examination of what other similar sites have in the way of policy,
- widespread involvement of users, management and Systems Administrators in the development of policy,
- acceptance of policy by management, and
- checking of the policy by lawyers.

Code of ethics

As the Systems Administrator on a UNIX system you have total control and freedom. All Systems Administrators should follow some form of ethical conduct. The following is a copy of the SAGE-AU Code of Ethical Conduct. The original version is available on the Web at http://www.sage-au.org.au/ethics.html.

SAGE-AU code of ethics

In a very short period of time computers have become fundamental to the organisation of societies world-wide; they are now entrenched at every level of human communication from government to the most personal. Computer systems today are not simply constructions of hardware -- rather, they are generated out of an intricate interrelationship between administrators, users, employers, other network sites, and the providers of software, hardware, and national and international communication networks.

The demands upon the people who administer these complex systems are wide-ranging. As members of that community of computer managers, and of the System Administrators' Guild of Australia (SAGE-AU), we have compiled a set of principles to clarify some of the ethical obligations and responsibilities undertaken by practitioners of this newly emergent profession.

We intend that this code will emphasise, both to others and to ourselves, that we are professionals who are resolved to uphold our ethical ideals and obligations. We are committed to maintaining the confidentiality and integrity of the computer systems we manage, for the benefit of all of those involved with them.

No single set of rules could apply to the enormous variety of situations and responsibilities that exist: while system administrators must always be guided by their own professional judgment, we hope that consideration of this code will help when difficulties arise.

(In this document, the term "users" refers to all people with authorised access to a computer system, including those such as employers, clients, and system staff.)

SAGE-AU code of ethics

As a member of SAGE-AU I will be guided by the following principles:

Fair Treatment

I will treat everyone fairly. I will not discriminate against anyone on grounds such as age, disability, gender, sexual orientation, religion, race, or national origin.

Privacy

I will access private information on computer systems only when it is necessary in the course of my duties. I will maintain the confidentiality of any information to which I may have access. I acknowledge statutory laws governing data privacy such as the Commonwealth Information Privacy Principles.

Communication

I will keep users informed about computing matters that may affect them -- such as conditions of acceptable use, sharing of common resources, maintenance of security, occurrence of system monitoring, and any relevant legal obligations.

System Integrity

I will strive to ensure the integrity of the systems for which I have responsibility, using all appropriate means -- such as regularly maintaining software and hardware; analysing levels of system performance and activity; and, as far as possible, preventing unauthorised use or access.

Cooperation

I will cooperate with and support my fellow computing professionals. I acknowledge the community responsibility that is fundamental to the integrity of local, national, and international network resources.

Honesty

I will be honest about my competence and will seek help when necessary. When my professional advice is sought, I will be impartial. I will avoid conflicts of interest; if they do arise I will declare them.

Education

I will continue to update and enhance my technical knowledge and management skills by training, study, and the sharing of information and experiences with my fellow professionals.

Social Responsibility

I will continue to enlarge my understanding of the social and legal issues that arise in computing environments, and I will communicate that understanding to others when appropriate. I will strive to ensure that policies and laws about computer systems are consistent with my ethical principles.

Workplace Quality

I will strive to achieve and maintain a safe, healthy, productive workplace for all users.

People skills

The ability to interact with people is an essential skill for Systems Administrators. The type of people the Systems Administrator must deal with includes users, management, other Systems Administrators and a variety of other people.

The following reading was first published in "The Australian Systems Administrator" (Vol 1, Issue 2, June/July 1994) the bimonthly newsletter of the Systems Administrators Guild of Australia (SAGE-AU). It provides an example of how a real-life System Administrator handles user liaison.

Communicating with Users

Next to balancing conflicting demands, communicating with users is the hardest part of my job. I tend to make a great effort for little gain, whereas in technical endeavours a little effort can produce a major, long-lasting improvement (for example, taking ten minutes to set up regular, automated scratch area cleanups has saved me hours of tedious work and the users a lot of frustration).

Also, with users there are emotions to take into account. It doesn't matter whether the computer respects you, but if the users respect you life is a lot easier.

My aim in communicating with users is to make life (my job and those of the users) easier by:

getting them to respect me (my judgment; my abilities; my integrity and professionalism).

teaching them all sorts of things, such as how to remove jobs from the printer queue; what *they* have to do to keep the systems secure; and when not to interrupt me with questions.

In this column I'm going to describe some of the communication vehicles I've tried, and how effective they've been for me. I'll start with those I've found least effective overall, and work my way up.

Probably the method most useless with the general user community is the **policy statement**. The typical user just isn't going to read it. However, it can be a good way of communicating with management. Drafting a good policy statement (based on discussions with everyone, but especially with them) shows you mean business and understand how your work fits into the organisation. It should cover the responsibilities of the systems administrator as well as those of the users.

Group meetings, whether of the users in general or of a committee of representatives, can help people -- again, especially senior people -- feel more confident that things are going OK, but aren't much use for disseminating information. If a meeting is run well you can have a productive discussion of major issues, but if run badly it is likely to turn into a gripe session.

Paper memos are to be avoided, because they encourage stiffness and formality. I use them only to answer other people's paper memos (which are usually complaints) and then only when I don't think the person will read it if I do it by email. Replying by email to a memo has the effect of saying "There's no need to be so formal".

There are a number of leading-the-horse-to-water methods, which only work if the user makes an effort. You can use **electronic information services**, such as bulletin boards, newsgroups, Gopher, or online manuals; and you can get together a library of printed **manuals and books**. If you provide easy access to

high-quality information, the interested user can learn a lot. Unfortunately it's often the disinterested user that you really want to reach.

People often **come to my office** to ask me things. You'd think that face-to-face communication would work the best, but in this particular setting it doesn't because I am not comfortable. It's not so much that I resent interruptions -- it's that I don't *have* an office, only a desk. There's no room for a visitor's chair; to talk to anyone I have to swivel round and face backwards; and people make a habit of sneaking up on me. Hopefully, one day my campaign for proper accommodation will be successful, and it will be interesting to see how much difference it makes.

Talking on the **phone** is only good for emergencies. *Someone* is always interrupted; there's no body language; and you tend to forget half of what you wanted to say.

I write a column, "Computer Corner", in our **staff newsletter**. I sometimes write about issues (such as what I'm trying to achieve) and sometimes about technical tips. This column isn't as useful as I'd hoped. The first problem is that there isn't room to say much, because the newsletter is short and a bit, shall we say, irregular. The second problem is that the rest of the newsletter tends to be kind of dull (lists of visitors; dry field-trip reports; the occasional births and deaths) so people aren't so eager to read it. When I pointed this out I was told that it is deliberately impersonal and non-funloving because some of the more senior readers are rather easily offended. Sigh.

Next on the scale are **signs** (on doors, noticeboards, etc) and electronic **messages-of-the-day**. People have a strong tendency to miss the former and ignore the latter. It may help to make them more interesting with graphics, pictures and human-interest items.

Seminars and workshops are worthwhile if you can get people to attend, but they're a lot of work. If not many turn up, you don't get much return on your investment. Students can sometimes be induced to attend by making it count towards their marks. In other situations, offering food, door prizes, alcohol, sex, drugs or rock-n-roll may help.

For explaining specific information (how to pick a good password; how UNIX file permissions work) I've found **paper handouts** reasonably effective. Some users take them quite seriously, even filing them for later reference. Unfortunately, others toss them straight in the bin.

After about 3 months in my current job I emailed everyone a **questionnaire**, asking such things as what they used the systems for, what new services they would like to see, and how often they did backups. I offered a chocolate frog to each person who replied. The subject line "Apply here for your FREE chocolate frog" caused some of the more pokerfaced members of staff to delete the mail without reading it, but otherwise the response was surprisingly good. In hindsight, I guess the questionnaire generated more PR than information, although it did confirm my suspicion that most people did not back up their data even though they were supposed to.

For me, the second most effective communication vehicle is **email**. Email is as informal as a personal visit or phone call, but you can get in a lot more information. It is also asynchronous: no-one has to be interrupted, and you don't have to wait for people to be available.

I often use **email broadcasts** for notification -- to tell people about impending downtime, for example. Email is quick, convenient, and reaches people who are working offsite. It is also informal and I think people feel more at ease with it than they do with paper memos and printed signs.

1-to-1 email gives people a sense of personal service without much of the hassle that normally entails. At my site people can email problem reports and questions to a special address, "computerhelp". Our stated aim is to respond within 2 working days. We don't always make it. But it does give people a point of contact at all times, even after hours, and it means we get a few less interruptions.

You'd think all of that might be enough, but no. My boss said, "You need to communicate more with the users, to tell them about what you're doing". I agreed with him. So I now produce a **fortnightly emailed bulletin**. It is longer and more formal than a typical email message, with headings and a table of contents. Most of the information in it is positive -- new software that we've installed, and updates on our program of systems improvements. I also include a brief greeting and a couple of witty quotations. Judging by the feedback I've received, this seems to be working remarkably well -- much better than the staff newsletter column.

The only thing that works better than email is **personal visits** where *I* am in *their* office, usually leaning over their screen showing them how to do something. Taking an interest in their work helps a lot. I find this easy where they are graphing the temperature of a lake in glorious colour, but more difficult where they are typing up letters. I don't do enough personal visiting, partly because I'm so busy and partly because I'm not keen on interrupting people. It usually happens only when they've asked a question that requires a "show me" approach.

A disadvantage of personal visits is that they help only one person at once, whereas with email you can reach all your users.

To sum up: in communicating with users, I aim to teach them things and get them to respect me. By sending email I can help the most people for the least effort, although personal visits have much more impact. There are other useful methods, such as policy statements, newsletters, handouts and seminars, but they may not reach the ones who need it most.

It's hard. Very hard. If you have any insights or ideas in this area, I'd love to hear them, and I'm sure the rest of the readers would too.

Communicating with management

Relationships between Systems Administrators and management can be tense generally because both sides don't understand the importance and problems of the other. Having good Systems Administrators is essential. As is having good management. Management is a difficult task which you won't understand or agree with until you have to perform it.

As a Systems Administrator you should keep in mind that the aims of management will not be the same as yours. Management is about profit. When you deal with management keep this in mind.

If you need an upgrade of a machine don't argue it on the basis that the load average is running at 5 and the disks are full. Argue it on the basis that due to

the lack of resources the sales force can't take orders and the secretaries are loosing documents which is leading to loss of customers.

Generally Systems Administrators tend to focus on achieving a good technical solution. This must be balanced with helping the company you are working for make money.

How not to communicate with users

The Bastard Operator from Hell is a classic (amongst Systems Administrators) collection of stories about a mythically terrible operator. It provides an extreme view of a bad system support person and is also quite funny (depending on your sense of humour). Some of the language may offend some people.

Bastard Operator from Hell

Available on the 85321 Web site under the Resource Materials section for week 1.

Conclusions

Systems Administration is a complex and interesting field requiring knowledge from most of the fields in computing. It provides a challenging and interesting career. The Linux operating system is an important and available alternative in the current operating systems marked and forms the practical component for this course.

Chapter 2

Information Sources and Problem Solving

Introduction

As a Systems Administrator you will be expected to fix any and all problems that occur with the computer systems under your control. There is no why that this book or 85321 can prepare you for every situation and problem you will come across. Additionally, for most of us mere mortals it is simply not possible for us to know everything that is required. Instead as a Systems Administrator must you know the important facts and be able to quickly discover any new information that you don't yet know. You must be able to diagnose and solve the problem even though you have never seen it before.

This chapter examines the sources of information that a Systems Administrator might find useful including professional associations,books, magazines, and the Internet. It also provides some guidelines about how you might go about solving problems you have never seen before. While some of this information is Linux specific most of it is applicable to any operating system.

As the semester progresses you should become familiar with and use most the information sources and the problem solving process presented here.

Other Resources

This chapter mentions a large number of Internet resources and includes the URLs. You can find an up to date listing of these links and other related links on the Links database on the 85321 Website (http://infocom.cqu.edu.au/85321/).

Other resources which are related to this chapter include

- Online lecture 2 on the 85321 website
- HOW-TOs
  Online Troubleshooting Resources HOW-TO, Reading List HOW-TO and the Staying Updated mini-HOW-TO

Information Sources

The following sections first examine the range of information sources you have available as a Systems Administrator. Each of the following section deals with a different type of resource. As a trainee Sys Admin you should find yourelf starting to use these resources during your study. Learning how to answer your own questions is perhaps the most important thing you can take from this text.

The information sources discussed in the following includes

- Professional associations
  There are quite a diverse list of professional assocations available which may be useful to a Systems Administrator.
- Books and magazines
  There are now a wide range of relevant books and magazines to chose from.
- Internet sources
  The Internet is perhaps the most comprehensive and up to date resources for Systems Administrators, if used correctly.

Professional organisations

Belonging to a professional organisation can offer a number of benefits including recognition of your abilities, opportunities to talk with other people in jobs similar to yours and a variety of other benefits. Most professional organisations distribute newsletters, hold conferences and many today have mailing lists and Web sites. All of these can help you perform your job.

Professional organisations a Systems Administrator might find interesting include

- Systems Administrators Guild of Australia (SAGE-AU, http://www.sage-au.org.au/),
- Systems Administrators Guild(SAGE) (the American version of SAGE-AU, http://www.usenix.org/sage/),
- Australian UNIX Users Group (AUUG, http://www.auug.org.au/),
- Australian Computer Society (ACS, http://www.acs.org.au/),
- Usenix (http://www.usenix.org.au/),
- Internet Society of Australia (http://www.isoc-au.org.au/)

This list has a distinct Australian, UNIX, Internet flavour with just a touch of the USA thrown in. If anyone from overseas or from other factions in the computer industry (i.e. Novell, Microsoft) has a professional organisation that should be added to this list please let me know (d.jones@cqu.edu.au).

The UNIX Guru Universe (UGU http://www.ugu.com/) is a Web site which provides a huge range of pointers to UNIX related material. It will be used throughout this chapter and in some of the other chapters in the text.

Professional Associations

The Resource Materials section on the 85321 Web site for week 1 has a page which contains links to professional associations and user organisations.

The SAGE groups

SAGE stands for Systems Administrators Guild and is the name taken on by a number of professional societies for Systems Administrators that developed

during the early 90s. There are national SAGE groups in the United States, Australia and the United Kingdom.

SAGE-AU

The Australian SAGE group was started in 1993. SAGE-AU holds an annual conference and distributes a bi-monthly newsletter. SAGE-AU is not restricted to UNIX Systems Administrators.

Both SAGE and SAGE-AU have a presence on the WWW. The Professional Associations page on the 85321 Web site contains pointers to both.

UNIX User groups

There are various UNIX user groups spread throughout the world. AUUG is the Australian UNIX Users Group and provides information of all types on both UNIX and Open Systems. Usenix was one of the first UNIX user groups anywhere and is based in the United States. The American SAGE group grew out of the Usenix Association.

Both Usenix (http://www.usenix.org/)and AUUG (http://www.auug.org.au/)have WWW sites. Both sites have copies of material from the associations' newsletters.

It should be noted that both user groups have gone beyond their original UNIX emphasis. This is especially true for Usenix which runs two important symposiums/conferences on Windows NT.

The ACS, ACM and IEEE

The ACS is the main professional computing society in Australia servicing people from all computing disciplines. The flavour of the ACS is much more business oriented than SAGE-AU.

The ACS is also moving towards some form of certification of computing professionals and some jobs may require ACS membership.

For more information refer to the WWW page (http://www.acs.org.au/).

The Association for Computing Machinery (the ACM, http://www.acm.org/) is one of the largest American professional computing societies. Its publications are considerably more technical and wide ranging than the ACS.

The Institute for Electrical and Electronics Engineers (IEEE, http://www.ieee.org/) also has a Computing Society (http://www.computer.org/).

Books and magazines

When a new computing person asks a technical question a common response will be RTFM. RTFM stands for **R**ead **T**he **F**ine (and other words starting with f) **M**anual and implies that the person asking the question should go away and look at documentation for the answer.

Around five years ago RTFM for a Systems Administrator meant reading the on-line man pages, some badly written manual from the vendor or maybe, if

lucky, a Usenet newsgroup or two. Trying to find a book that explained how to use `cron` or how to set up NFS was a difficult task.

Since then there has been an explosion in the number of books and magazines that cover Systems Administration and related fields. This is especially noticable over the last couple of years with the huge hype surrounding Linux and Open Source software. The following pages contain pointers to a number of different bibliographies that list books that may be useful.

A Linux specific "magazine" which anyone with access to the 85321 CD-ROM/Website (or to the Linux Documentation Project) can read is the Linux Gazzette.

Bibliographies

UNIX, Systems Administration and related books.

The Resource Materials section for week 1, on the 85321 Web site and CD-ROM, has a collection of pointers to books useful for 85321 and Systems Administrators in general.

O'Reilly books

Over the last few years there has been an increase in the number of publishers producing UNIX, Systems Administration and network related texts. However one publisher has been in this game for quite some time and has earned a deserved reputation for producing quality books.

A standard component of the personal library for many Systems Administrators is a collection of O'Reilly books. For more information have a look at the O'Reilly Web site (http://www.ora.com/).

Magazines

There are now a wide range of magazines dealing with all sorts of Systems Administration related issues, including many covering Windows NT.

Magazines

The 85321 Web site contains pointers to related magazines under the Resource Materials section for week 1.

Internet resources

The Internet is by far the largest repository of information for computing people today. This is especially true when it comes to UNIX and Linux related material. UNIX was an essential part of the development of the Internet, while Linux could not have been developed without the ease of communication made possible by the Internet. If you have a question, a problem, need an update for some software, want a complete operating system or just want to have a laugh

the Internet should be one of the first places you look as a Systems Administrator.

So what is out there that could be of use to you? You can find

- software
- discussion forums, and
- information.

Each of these is introduced in more detail in the following sections.

The 85321 Website

The 85321 website (http://infocom.cqu.edu.au/85321/) contains mirrors, pointers and other resources related to some of the Internet related material discussed below. These resources will include

- a link database
  Chances are the links listed here will become out of date. The 85321 Website maintains a database of Linux and Systems Administration related links which include ratings. You can add links as to this database.
- a FAQ database
  The LDP and most newsgroups maintain lists of common questions (with answers) this database allows you to quickly search and view these questions.
- a Deja search interface
  The 85321 home page contains a form where you can perform a search of DejaNews.

How to use the Internet

By this stage it is assumed that you should be a fairly competent user of the Internet, the World-Wide Web, email, Usenet news and other net based resources. If you are a little rusty or haven''t been introduced to many of these tools there are a large number of tutorials on the Internet that provide a good introduction. A good list of these tutorials is held on the Yahoo site (http://www.yahoo.com/).

Software on the Internet

There is a large amount of "free" UNIX software available on the Internet. It should be remembered that no software is free. You may not pay anything to get the software but you still have to take the time to install it, learn how to use it and maintain it. Time is money.

GNU software (GNU is an acronym that stands for GNU's Not UNIX) is probably the best known "public-domain" software on the Internet. Much of the software, for example `ls` `cd` and the other basic commands, that comes with Linux is GNU software. In fact there is a trend amongst some people to call Linux, GNU/Linux, to reflect the amount of GNU software a Linux distribution uses.

The Gnu Manifesto

A copy of the GNU manifesto is available on the 85321 Web site and CD-ROM under the Resource Materials section for this week.

The GNU website (http://www.gnu.org/) contains a lot more information about GNU's projects.

A good place to go to get the latest Open Source software (if you are based in Australia) is the AARNet mirror (http://www.mirror.aarnet.edu.au or ftp://mirror.aarnet.edu.au). It contains mirrors of a lot of popular Open Source software including GIMP, MySQL, Perl, Linux, Apache, KDE etc.

Discussion forums

Probably the biggest advantage the Internet provides is the ability for you to communicate with other people who are doing the same task. Systems Administration is often a lonely task where you are one of the few people, or the only one, doing the task. The ability to share the experience and knowledge of other people is a big benefit.

Major discussion forums on the net include

- Usenet news
- Mailing lists
- other discussion tools

Usenet news

Once one of the most popular features of the Internet Usenet news has lost some of its popularity as the number of people on the Internet increases. Usenet news is a collection of discussion forums which are distributed and read with specialised software (i.e. You read Usenet news with a news reader). There are discussion forums on a wide range of topics from purely social through to very technical.

A Systems Administrators technical interested is usually attracted to the large number of Linux/Systems Administration/Network related newsgroups. Newsgroups are a good place to ask questions, listen to other people and learn about topics. Some of the more useful newsgroups for this 85321 include

- `comp.os.linux.*`
  There are a large number of newsgroups under this heading discussing most Linux related topics, e.g. `comp.os.linux.setup` is used for discussion about installing and setting up a Linux box.
- `comp.unix.*`
  Another large collection of newsgroups talking about UNIX in particular.

Useful groups include `comp.unix.questions` for general UNIX questions and `comp.unix.admin` for Systems Administration type questions.

- `aus.computer.linux`
  An Australian Linux newsgroup.

http://www.linuxresources.com/online.html maintains a more detailed description and list of Linux newsgroups.

Just the FAQs

As you might imagine there are some questions which are asked again and again and again on newsgroups. Rather than repeat the answers to these questions again and again most newsgroups maintain a list of Frequently Asked Questions (FAQs). It is considered good practice when joining a newsgroup for the first time to read the the FAQs. It is a compulsory task before you ask a question on a newsgroup.

You can access the FAQs for most newsgroups at http://www.faqs.org/. This site contains over 3300 seperate FAQs written by over 1250 authors covering 1700 newsgroups. The 85321 website also maintains a collection of many of the FAQs relevant to Systems Administration.

Exercises

2.1. There is a newsgroup called `comp.os.unix`. Like many newsgroups this group maintains an FAQ. Obtain the `comp.unix.questions` FAQ and answer the following questions
- find out what the `rc` stands for when used in filenames such as `.cshrc` `/etc/rc.d/rc.inet1`
- find out about the origins of the `GCOS` field in the `/etc/passwd` file

Google and Deja News

Google (http://www.google.com/) and DejaNews (http://www.deja.com/) are among the two most useful websites when it comes to finding technical information. Learn to use and enjoy them. Google is a Web search engine which uses technology to provide better ranking of Websites than other search engines.

DejaNews is an archive and search engine of posts to Usenet News. The quickest way to find solutions to a lot of problems you will come across is to perform an appropriate search on DejaNews.

Learn to use these sites.

Mailing lists

For many people the quality of Usenet News has been declining as more and more people start using it. One of the common complaints is the high level of beginners and the high level of noise. Many experienced people are moving towards mailing lists as their primary source of information since they often

are more focused and have a “better” collection of subscribers and contributors.

Mailing lists are also used by a number of different folk to distribute information. For example, vendors such as Sun and Hewlett Packard maintain mailing lists specific to their operating systems (Solaris and HP-UX). Professional associations such as SAGE-AU and SAGE also maintain mailing lists for specific purposes. In fact, many people believe the SAGE-AU mailing list to be the one of the best reasons for joining SAGE-AU as requests for assistance on this list are often answered within a few hours (or less).

Mailing lists

One good guide to all the mailing lists that are available is Liszt, mailing list directory (http://www.liszt.com/).

The UNIX Guru’s Universe also maintains a directory of mailing lists related to Sys Admin.

Other Discussion Forums

There are also other forums that may be useful for Systems Administrators and make use of technology other than Usenet news or mailing lists. These forums often use IRC or Web-based chat and bulletin board facilities.

Over the last year or so Web-based bulletin board like systems have come to the fore. Examples include

- Slashdot, http://www.slashdot.org/
  A bunch of people contribute links and information about what is happening in the nerd world. Includes a lot of interesting Linux related material.
- Linux Today, http://www.linuxtoday.com/
  A slightly more business oriented version of Slashdot. A bit more serious but still a great information source.
- Freshmeat, http://www.freshmeat.net/
  A place where people announce the latest releases of Open Source software

Internet based Linux resources

Linux would not have been possible without the Internet. The net provided the communications medium by which programmers from around the world could collaborate and work together to produce Linux. Consequently there is a huge collection of Internet based resources for Linux.

The Linux Documentation Project

The best place to start is the Linux Documentation Project (LDP). The aim of this project is to produce quality documentation to support the Linux community. The original LDP page is located at http://www.linuxdoc.org/

A mirror of the LDP pages is maintained on the 85321 Web site and a copy of these pages can be found on the 85321 CD-ROM.

A major source of information which the LDP provides are the HOW-TOs. HOW-TOs are documents which explain how to perform specific tasks as diverse as how to install and use StarOffice (a commercial office suite that is available free, and may well be on the 85321 CD-ROM) through to detailed information about how the Linux boot-prompt works.

The HOW-TOs should be the first place you look for specific Linux information. Copies are available from the LDP Web pages.

RedHat

This version of the text is written as a companion for RedHat Linux. As a result it will be a common requirement for you find out information specific to RedHat Linux. The best source on the Internet for this information is the RedHat site, http://www.redhat.com/.

Possibly the most important information source on the RedHat site are the updates/errata pages for the distribution of RedHat Linux you have installed. There will be errors and new software for RedHat Linux and RedHat are the best source. If you installed RedHat off a CD you may also find some of the updates and errata on it.

Additionally RedHat provide three manuals with RedHat Linux 6.1. All three of the manuals are available as postscript files on the RedHat Linux CD. The three manuals are

- Red Hat Installation Guide
  A copy of this will be included in the package received by distance education students of CQU. This guide is designed to help you install RedHat Linux 6.1 onto your computer.
- Red Hat Linux Getting Started Guide
  Provides information you will likely need to configure your Linux box including an overview of Gnome. Gnome is the default desktop environment with RedHat Linux 6.1.
- RedHat Linux Reference Guide
  Contains references to a wide range of information you may need while using your computer.

Problem Solving

I will guarantee that you will have problems with Linux while you are attempting the tasks and reading about the concepts introduced in this text. Most of the time these problems will be unlike anything you have ever seen before. You probably won't have any idea what the actually problem is. You may have difficulty even starting to describe it.

Don't PANIC!!!!

Every Systems Administrator, every computer user has faced this same problem. The difference between a bad Systems Administrator and a good one is the ability to problem solve. If you can learn to solve problems you have never faced before then you can do anything.

Guidelines for solving problems

The "Linux Installation and Getting Started Guide" (part of the Linux Documentation Project) provides the following guidelines for solving problems. They are a good start.

Remain calm.
Never ever attempt to solve anything computer related while you are upset, angry, sad, tired or emotional in any way. Any of these feelings will adversely affect your ability to solve the problem. In other words they are likely to make things worse.

Learn to appreciate self-reliance
The key to problem solving is that you have to do it. If you are continually relying on other people to solve your problems you will never learn anything. Try and solve your problems first. If you can't solve your problems talk to someone about what you are doing and why you are doing it. Try and figure out why you can't solve problems. The idea is that you have to do this before you can start problem solving on your own.

Consult all available sources of information
Most people, especially in the Linux world, are more than happy to help out with a problem. However, they can get very upset if the question you are asking has already been answered in some documentation or on the Web.

Know the best places to look
There are a huge number of different places from which you can get information about Linux and Systems Administration. The same applies to Windows NT. If you know which sites are best for which sort of information you can significantly cut down your search time.

Refrain from asking spurious questions
Linux has the best support mechanism of any operating system and the best thing about it is that it is free. However, there are accepted codes of behaviour and one of them is "Don't ask stupid questions".

When asking for help be polite, terse and informative
"I can't boot my Linux computer" is not informative. If you don't provide sufficient information no-one will be able to offer help. At the other extreme, if you provide too much information people won't be bothered to read it. Lastly, no-one likes a rude person. If you are rude people are likely to be rude back again and not help.

Contribute to the community
The number of people contributing to Linux newsgroups, the LDP and other areas is the main reason why Linux support is so good. If you know the answer to a problem share it with other people and help make the support even better.

These guidelines are a good start. The following are some additional ones I've developed over the last few years.

- Understand what the problem is.
  This may seem to be a bit obvious. However, 5 years of teaching 85321 has shown that the most common reason students can't solve their problems

is that they don't understand what the problem is. If you don't know what the problem is find out or following the next suggestion.

- Break the problem down into smaller steps
  If the problem is too complex, too large or you don't understand it try breaking it down into smaller steps so you can solve (or understand) those.
- Don't get to close, walk away
  There will be times when the best thing you can do is walk away and do something else for a while. Any person with experience in the computing field should be able to tell a story about how a solution to a frustrating problem popped into their head while mowing the lawn, watching television or having a shower. Some distance from the problem can provide the bit of perspective you needed to figure out the problem.
- Talk to people.
  A group of people will always be able to solve more problems than a single person. More people means more experience and different perspectives. If you don't know the solution chances are someone else does.
- Record the solution
  This could be considered as contributing to the community but it also serves a much more selfish reason. If you have faced the problem once then chances are it you, or another Systems Administrator, will face it again. If you have documented the solution to the problem then solving it the 2nd, 3rd or 4th time will be much quicker.

Examples of solving problems

The 2nd 85321 online lecture includes three examples of how to use the resources and approaches introduced above to solve problems.

Conclusions

A lot of time spent by Systems Administrators is consumed attempting to solve new problems. A Systems Administrator must be able to solve new problems. Being aware of the available information sources and their relative merits is an important step towards being able to solve these problems. While the information presented in this chapter may be useful it is experience which is the most useful tool in solving problems. Take the time to gain this experience.

Review Questions

2.1 Use the information sources here to solve some of the problems you are currently having. If you aren't having problems, tind a question from one of the Linux or UNIX newsgroups.

2.2. Examine the errata list for your version of RedHat Linux. Do any of these errata appear important to your system?

Chapter 3
Using UNIX

Introduction

A Systems Administrator not only has to look after the computers and the operating system, they also have to be the expert user (or at least a very knowledgeable user) of their systems. When other users have problems where do they go? The documentation? Online help facilities? No, they usually go to the Systems Administrator. Adding to the importance of the material covered in the next few chapters is that a number of the topics introduced here are the foundations on which some of the more complex Systems Administration topics are built. If you don't understand these concepts now you will have problems later on.

The following reading aims to start you on the road to becoming an expert UNIX user. Becoming a UNIX guru can only be achieved through a great deal of experience so it is important that you spend time using the commands introduced in this chapter.

Other Resources

Resources explaining the basics about using Linux and UNIX are quite numerous. Some of the other resources which mention similar concepts to this chapter include

- The RedHat Manuals
  RedHat 6.1 comes with three manuals. This chapter refers to some of these manuals as a source for more information.
- Online lecture 4
  Included on the 85321 website/CD-ROM this lecture with slides and audio covers complementary material to this chapter.
- Linux Installation and Getting Started Guide
  One of the guides included with the Linux Documentation Project includes some basic information. A copy of the LDP is available on the 85321 website/CD-ROM.

What you need to learn

All of the concepts and material you will use to learn how to use Linux is not included in this book. Throughout this chapter you will be referred a collection of documents and web pages. It is important that you actually read this material and more importantly you should attempt to practice the commands and practices you learn about. If you don't use it you lose it.

In order to be able to really understand the material introduced later in this chapter and to be able to perform the required tasks with a minimum of effort you MUST become familiar with the following

- How to get around the Linux file hierarchy.
  Using the GUI "explorer-like" tools provided by Gnome and KDE are not sufficient. You must be familiar with using commands like cd ls mkdir rm ls cp
- How to get the most out of the user interface you are provided with.
  Making use of the GUI, while not a replacement for the command line, will make some task easier.
- Be able to use simple command line tools
  I'm repeating it, just in case you ignored it the first time. You MUST BE ABLE to use the simple UNIX commands such as ls cd, mkdir rm ls cp etc
- Be able to use the `vi` editor.
  As with the UNIX command line many of you will question why you need to use `vi`. The simple reason is that it will make your life much easier later in the semester if you learn how to use vi now.
- Understand the Linux file permissions system.
  This is especially essential. An understanding of the Linux file permissions system is an absolute necessity for when you move onto the more complex Systems Administration concepts introduced in later chapters. If you don't understand this now you will have major problems later on.
- Be able to manipulate and view the processes currently running.
  As with file permissions the ability to manipulate and view the processes on a Linux box is an essential skill for a Systems Administrator.

Introductory UNIX

Five years ago this section used to be quite easy for the majority of people reading this text. However, since then the explosion in Windows and other GUIs means that most people have little or no experience with using the command line. Some of you may not even now what the command line is!!!

The command line is a name given to the text-based interface which was common a few years ago and is still present in the form of the MS-DOS prompt in the Windows world and command-line shells in the UNIX world. This interface uses a process something like this

- computer displays a prompt
- use types a command, usually in the format
  command_name a list of parameters
- computer tries to carry out that command and displays any output
- computer displays a prompt again (and we loop back to the top)

I'm sorry to say but as a Systems Administrator you have to know how to use the command line. This means you have to forget about that nice GUI

provided by Explorer and start to understand the structure of files and directories used by Linux.

Why do I need to know the command line?

Some possible answers to this question include

- The GUI isn't always available.
  Unlike Windows NT a GUI is not a compulsory part of UNIX. This is one of the reasons why a small 386 running Linux can act as the server for a small organisation. It also means that there will be times as a Systems Administrator that you will have to perform tasks without a GUI. Those times are generally when something has broken. You won't have the time to learn how to use the command-line then, take the time to do it now.
- The command line is often more efficient and powerful.
  There are a wide number of tasks which you will have to perform in your computing career that are not suited to using a GUI. These tasks can be done quicker and easier using the command line.

The last reason for those of you studying 85321 is that your ability to use the command line is assessable. If you don't know how to use it you will lose marks.

How do I learn all this stuff?

The simple answer is practice.

You can't learn this material without experience. First you need to read and listen to the material pointed to below. Then you must take the time to perform the tasks set and also possible a number of others until you are comfortable with using the command line.

Taking the time to do this now will save you time later.

Basic UNIX

There is a wide range of material on the Internet which will introduce you to the basics of using UNIX. The following is a list of some of those available from the 85321 Website/CD-ROM. Please use the resources which best suit you

7.1. Online Lecture 2
Produced for the 1999 offering of 85321 this lecture covers using the command line, vi and the file hierarchy

7.2. The Linux Installation and Getting Started Guide
This guide was produced as part of the Linux Documentation Project (a mirror of this project is included on the 85321 Website/CD-ROM). It has a Linux tutorial which covers much of the basic material

7.3. The Red Hat Linux Getting Started Guide
Produced by Red Hat and included on the 85321 Website/CD-ROM in PDF and HTML formats this guide also covers much of the introductory material you will need.

Exercises

3.1. What UNIX commands would you use to
- change to your home directory
- display the list of files in the current directory
- display `my name is fred` onto the screen
- copy the file `tmp.dat` from the current directory to the directory `data` underneath your home directory and after the file has been copied delete it

3.2. What will the following UNIX commands do? ***Don't execute a UNIX command if you aren't sure what it is going to do.*** In particular do not try to execute the first command below.
```
rmdir ~
cat /etc/passwd
ls ../../fred/doc/tmp
```

3.3. Indicate which of the following paths are full or relative
a./root/ b.../root/ c./usr/../root d./home/david/

3.4. Assuming you are currently in the /home/david/tmp/ directory write the full path of your final location if you perform the following commands
a.cd ../85321/ b.cd /usr/lib c.cd ~/85321

3.5. Answer the following questions
a.Where would you find the home directory for the root user?
b.Where would you store some temporary files?
c.Where do you normally find the home directories of "normal" users?
d.Assuming you were currently in the directory containing the boot configuration files, how would you change into the directory containing the system configuration files and scripts?

UNIX Commands are programs

The UNIX commands that have been introduced so far are stored on a UNIX computer as executable files. All the commands on a UNIX systems are either stored on the hard-drive as executable files or are understood by a shell (more on these in a later chapter).

Most of the standard commands will be stored in standard binary directories such as `/bin /usr/bin /usr/local/bin`. On my system running RedHat version 6.1 there are 1494 different files in the directories `/bin`, `/usr/bin` and `/usr/sbin`. Which means over 1494 different commands (or there abouts).

vi

A major task of any user of a computer is editing text files. For a Systems Administrator of a UNIX system manipulation of text files is a common task due to many of the system configuration files being text files. The most common, screen-based UNIX editor is vi. The mention of `vi` sends shudders through the spines of some people, while other people love it with a passion. `vi is difficult to learn, however it is also an extremely powerful editor which can save a Systems Administrator a great deal of time.`

As you progress through this subject you will need an editor. `vi` is an anachronistic antique of an editor hated by most people. So why should you use it? Reasons include

- it is very powerful,
  How many editors do you know that can take the first 20 characters of every line in a file and swap them with the second set of 20 characters (something I've had to do)
- it is the only screen editor available on every UNIX system
- There will be times when a Systems Administrator cannot use a full screen editor. At times like this you must resort to single line editors like **`ed`** and **`ex`**. **`vi`** grew out of the `ex` editor and so uses many of the same commands. Learning and using these commands in **`vi`** can save you time later on.

As a result of all this it is strongly suggested that you use `vi` wherever possible in studying for this unit. Early on you will find using `vi` a hassle but sticking with it will be worthwhile in the end.

An introduction to `vi`

Most people when confronted with vi for the first time are put off by its completely foreign nature and lack of any queues about what to do. vi actually uses a very simple "model of operation". Central to this is the fact that vi is a modal editor. This means vi has a number of different modes and the same action can have completely different meaning in different modes.

vi modes

vi can be said to have three modes

- command mode
  This is the default mode vi is in when you start it up. In this mode most of the keys on the keyboard perform vi commands. For example, hitting the e key during vi command mode moves the cursor onto the next word. Use the list of vi commands in any of the vi command references discussed below to find out more.
- insert mode
  This is the mode in which vi behaves most like other editors. If you hit the k key it will insert k into the current location of the cursor and move the cursor on.

- ex mode
  In ex (sometimes called colon mode) you get to access a range of commands from the ex editor (and you thought vi was hard to use). A common one you will use is :wq which writes/saves the current file and then quits vi (wq).

vi Transitions

Knowing about the vi modes is no good unless you know how to go from one mode to another. For example, you can't actually type anything into a text file you are creating without knowing how to go from command mode to insert mode. Common transitions include

- command to insert
  A number of vi commands take you from command to insert modes (e.g. i o O)
- insert to command
  You'll do this transition when you want to save a file (usually). Hitting the ESC key is usually enough to achieve this
- command to ex
  Simply hitting the : (colon) key will put you into ex mode. You will know this because the colon will appear at the bottom of the screen.
- ex to command
  Simply hitting enter at the end of an ex command takes you back into command mode.

Using vi

The section for week 2 in the Link database on the 85321 Web page contains a number of resources which introduce you to vi. This includes the 4th online lecture which has a number of slides and examples of using vi.

vi, vim and ^Ms

A common problem in the last couple of years for 85321 students has shown itself up as shell scripts which can't run (you'll be getting to shell scripts in a couple of weeks). The problem usually occurs when the student copies a text file created under Windows to Linux. The cause of the problem is that UNIX and Microsoft indicate the end of the line in different ways

- carriage return, line-feed
  Used by Microsoft operating systems
- line feed
  Use by Linux.

The extra character, the carriage return, causes problems in some situations, e.g. When you want to run the text file as a shell script.

The solution is to remove the extra characters. One method is to use vi. The trouble is that by default vim, the version of vi on Linux, is smart enough to hide the carriage returns.

If you have a text file, which has carriage returns in it, and you want to see the carriage returns you have to do the following

```
vi -b filename
```

The -b causes vi to work in binary mode and you will now be able to see the carriage returns which look like ^M and appear at the end of the line. Carriage return is actually one character. ^M is the standard UNIX way of representing a single control character. If you try to delete the ^M with the x command you will find that there is only on character.

To delete all the carriage returns in a file you can use the following command

```
1,$s/^M//g
```

Where you don't type the ^ character and then the M character. Instead you hold the CONTROL key down and hit the c key and then hit the m key. What this command does should become clear when we talk about regular expressions in a later chapter.

UNIX commands

A UNIX system comes with hundreds of executable commands and programs . Typically each of these programs carries out a particular job and will usually have some obscure and obtuse name that means nothing to the uninitiated. That said the names of most of these commands actually do make some sort of sense once you have a bit of knowledge.

In the following you are introduced to the philosophy and format of UNIX commands. It is also emphasised that there is almost always going to be a UNIX command (or a combination of them) to perform the task you wish to accomplish. You need to become familiar with how to find out about the available commands.

Philosophy of UNIX commands

There are no set rules about UNIX commands however there is a UNIX philosophy that is used by many of the commands.

- small is beautiful,
  UNIX provides the mechanisms to join commands together so commands should do one thing well.
- 10 percent of the work solves 90 percent of the problems,
  UNIX was never designed to solve all problems, it was designed to solve most requirements without too much hassle on the programmer's part.
- solve the problem, not the machine,
  Commands should ignore any machine specific information and be portable.

- solve at the right level, and you will only have to do it once.
  The key to UNIX problem solving is only to do it once e.g. pattern matching is only implemented once, in the shell, not in every command.

One of the central tenants of the UNIX command philosophy is to provide a flexible, adaptable toolbox approach to solving problems. The idea is not to provide a single large program which does everything. Instead you have small, purpose built commands which can be easily combined to perform much larger tasks. An evolution rather than creation approach to solving problems.

UNIX command format

UNIX commands all use the following format

```
command_name -switches parameter_list
```

Component	Explanation
command_name	the name of the actual command, generally this is the name of the executable program that is the command
-switches	The - symbol is used to indicate a switch. A switch modifies the operation of a command.
parameter_list	the list of parameters (or arguments) that the command will operate on, could be 0, 1 or more parameters, parameters are separated by white space characters (space, TAB)

Table 3.1
UNIX command format

Please note: there must be spaces between each component of a UNIX command line. Under MS-DOS it was possible to perform commands like

- cd/dos
  Change the current directory into the /dos directory.

Under UNIX this command will be interpreted as run the command cd/dos. This means UNIX will normally try to find an executable file called cd/dos. The UNIX shell (the command which interprets the command line and tries to execute commands) uses the space character to tell the difference between the different components of the command line. Under UNIX the command would have to be

- cd /dos

Example commands

- `ls -l`
  The switch `-l` is used to modify the action of the `ls` command so that it displays a long listing of each file.
- `ls -l /etc/passwd / /var`
  Commands can take multiple parameters.
- `ls -ld /etc/passwd / /var`
  Multiple switches can also be used.

Linux commands take multiple arguments

Unlike MS-DOS, UNIX commands can take multiple arguments.

However, the multiple parameters must be separated by space characters.

Exercises

3.6. One of your users has created a file called `-tmp`? (The command `cat /etc/passwd > -tmp` will do it.) They want to get rid of it but can't. Why might the user have difficulty removing this file? How would you remove the file? You might have to refer to the online help section below to find the answer.

A command for everything

A fairly intelligent and experienced would be computer professional has just started using UNIX seriously (he was a student in the very first offering of this subject). He gets to a stage where he wants to change the name of some files.

Being an MS-DOS junkie from way back what command does he look for? The `rename` command of course. It doesn't work! "That's a bit silly!", he thinks, "You would think that UNIX would have a `rename` command."

It just so happens that this person has just completed a C programming subject in which one of the assignments was to write a `rename` command. So he spends the next day trying to write and compile this program. After much toil and trouble he succeeds and follows good administration policy and informs all the other students of this brand new wonderful program he has written. He goes into great detail on how to use the command and all the nice features it includes.

They all write back to tell him about the UNIX command `mv` (the move command) that is the UNIX command that is equivalent to `rename`.

The moral of the story

The moral of this story is that if you want to do something under UNIX, then chances are that there is already a command to do it. All you have to do is work out what it is.

Online help

UNIX comes with online help called **man pages**. Man pages are short references for commands and files on a UNIX system. They are not designed as a means by which newcomers to UNIX can learn the commands. Instead they are a reference to the command to be used by someone who understands the basics of UNIX and UNIX commands.

The man pages are divided into different sections. Table 3.2 shows the sections that Linux uses. Different versions of Linux use slightly different sections.

Section number	Contents
1	user commands
2	system calls
3	Library functions
3c	standard C library
3s	standard I/O library
3m	arithmetic library
3f	Fortran library
3x	special libraries
4	special files
5	file formats
6	games
7	miscellaneous
8	administration and privileged commands

Table 3.2
Manual Page Sections

Using the manual pages

To examine the manual page for a particular command or file you use the `man` command. For example if you wanted to examine the man page for the `man` command you would execute the command `man man`.

Is there a man page for...

The command `man -k` *`keyword`* will search for all the manual pages that contain *`keyword`* in its synopsis. The commands `whatis` and `apropos` perform similar tasks.

Rather than search through all the manual pages Linux maintains a keyword database in the file `/usr/man/whatis`. If at any stage you add new manual pages you should rebuild this database using the `makewhatis` command.

The -K switch for the man command forces it to search through all of the manual page for the work. You should realise that with the size and number of manual pages this operation can take quite a while.

If there is a file you wish to find out the purpose for you might want to try the `-f` option of the `man` command.

man page format

Each manual page is stored in its own file formatted (under Linux) using the `groff` command (which is the GNU version of nroff). The files can be located

in a number of different directories with the main manual pages located under the `/usr/man` directory.

Under `/usr/man` you will find directories with names `man`*n* and `cat`*n*. The *n* is a number that indicates the section of the manual. The files in the `man` directories contain the `groff` input for each manual page. The files in the `cat` directories contain the output of the `groff` command for each manual page.

Generally when you use the `man` command the `groff` input is formatted and displayed on the screen. If space permits the output will be written into the appropriate `cat` directory.

3.7. What commands would you use to do the following
1. View mkdir(2)
2. Find the command to print a file
3. How many different manual pages exist for mkdir
4. Describe the contents of section 8 of the manual pages.

HTML versions of Manual Pages

The 85321 website contains a manual page section which contains a collection of manual pages in HTML format.

Some UNIX commands

There are simply too many UNIX commands for this chapter to introduce all, or even most of them. The aim of the following is to show you some of the basic commands that are available. To find the remainder you will have to discover them for yourself. One method for becoming more familiar with the available commands is to

- look at the filenames in the `/bin /usr/bin /usr/local/bin` directories,
  These are the "binary" directories which contain the executable programs which are the UNIX commands.
- take the filename and look at the manual page
  Each of the commands will have a manual page which will explain what the command does and how you can use it.

The commands introduced in this table can be divided into categories based on their purpose

- identification commands,
  These commands identify the user or the system.
- simple commands,
  Perform some simple, non-specific task.
- filters.
  Filters take some input, modify it and then output it.

Command	Purpose	Command	Purpose
`date`	Display the current time and date	`who`	display who is currently on the computer
`banner`	Display a large banner	`cal`	display a calendar
`whoami`	Displays your current username	`cat`	display the contents of a file
`more` and `less`	Display the contents of a file a page at a time	`head`	display the first few lines of a file
`tail`	Display the last few lines of a file	`sort`	sort the content of a file into order
`uniq`	Remove duplicate lines from a file	`cut`	remove columns of characters from a file
`paste`	join columns of files together	`tr`	translate specific characters
`grep`	Display all lines in a file containing a patter	`wc`	count the number of characters, words and lines in a file

Table 3.3
Basic UNIX commands

Identification Commands

who

Displays a list of people currently logged onto the computer.

```
dinbig:/$ who
david    tty1     Feb  5 14:27
```

whoami

Displays who the computer thinks you are currently logged in as.

```
dinbig:/$ whoami
david
```

uname

Displays information about the operating system and the computer on which it is running

```
[david@beldin david]$ uname
Linux
[david@beldin david]$ uname –a
Linux beldin.cqu.edu.au 2.0.31 #1 Sun Nov 9 21:45:23 EST 1997 i586
unknown
```

Simple commands

The following commands are simple commands that perform one particular job that might be of use to you at some stage. There are many others you'll make use of.

Only simple examples of the commands will be shown below. Many of these commands have extra switches that allow them to perform other tasks. You will have to refer to the manual pages for the commands to find out this information.

date

Displays the current date and time according to the computer.

```
dinbig:/$ date
Thu Feb  8 16:57:05 EST 1996
```

banner

Creates a banner with supplied text.

```
dinbig:/$ banner -w30 a
          ##
        ######  ##
       ##    ## ###
        #    #     #
        ##  ##   ##
       ############
       ##
```

cal

Display a calendar for a specific month. (The Linux version might not work).

```
bash$ cal 1 1996
   January 1996
 S  M Tu  W Th  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

Filters

Filters are UNIX commands that take input or the contents of a file, modify that content and then display the result on output. Later on in this chapter you will be shown how you can combine these filters together to manipulate text.

`cat`

The simplest filter. `cat` doesn't perform any modification on the information passed through it.

```
bash$ cat /etc/motd
Linux 1.2.13.
```

`more` and `less`

These filters display their input one page at a time. At the end of each page they pause and wait for the user to hit a key. `less` is a more complex filter and supports a number of other features. Refer to the `man` page for the commands for more information.

`head` and `tail`

`head` and `tail` allow you to view the first few lines or the last few lines of a file.

Examples

- `head chap1.html`
  Display the first 10 lines of chap1.html
- `tail chap1.html`
  display the last 10 lines of chap1.html
- `head -c 100 chap1.html`
  display the first 100 bytes of chap1.html
- `head -n 50 chap1.html`
  display the first 50 lines of chap1.html
- `tail -c 95 chap1.html`
  display the last 100 bytes of chap1.html `sort`

`sort`

The `sort` command is used to sort data using a number of different criteria outlined in the following table.

Switch	Result
`-r`	sort in descending order (default is ascending)
`-n`	sort as numbers (default is as ASCII characters) When sorting numbers as numbers 100 is greater than 5. When sorting them as characters 5 is greater than 100.
`-u`	eliminate duplicate lines
`+`*`number`*`n`	skip *`number`* fields
`-t`*`character`*	specify *`character`* as the field delimiter

Table 3.4
Switches for the sort command

Examples

The following examples all work with the `/etc/passwd` file. `/etc/passwd` is the file that stores information about all the users of a UNIX machine. It is a text file with each line divided into 7 fields. Each field is separated by a `:` character. Use the `cat` command to view the contents of the file.

- `sort /etc/passwd`
  sort in order based on the whole line
- `sort -r /etc/passwd`
  reverse the order of the sort
- `sort +2n -t: /etc/passwd`
  sort on third field, where field delimiter is `:` (skip the first two fields)
- `sort +2n -t: -n /etc/passwd`
  same sort but treat the field as numbers not ASCII characters

`uniq`

`uniq` is used to find or remove and duplicate lines from a file and display what is left onto the screen. A duplicate to `uniq` is where **consecutive** lines match exactly. `sort` is often used to get the duplicate lines in a file into consecutive order before passing it to `uniq`. Passing a file from one command to another is achieved using I/O redirection which is explained in a later chapter.

Examples

- `uniq names`
  remove duplicate lines from `names` and display them on the screen
- `uniq names uniq.names`
  remove duplicates lines from `names` and put them into `uniq.names`
- `uniq -d names`
  display all duplicate lines

`tr`

Used to translate specified characters into other characters. `tr` is used in conjunction with I/O redirection which is explained in the next chapter. In the examples below the < character is an I/O redirection character.

Examples

- `tr a z < /etc/passwd`
  translate all a's to `z's` in `/etc/passwd` and display on the screen
- `tr '[A-Z]' '[a-z]' < /etc/passwd`
  translate any character in between `A-Z` into the equivalent character between `a-z`. (make all upper-case characters lower case)
- `tr -d ' ' < /etc/passwd`
  delete any single space characters from the file

`cut`

Is used to "cut out" fields from a file. Try `cut -c5-10 /etc/passwd`. This will display all the characters between the 5th and 10th on every line of the file `/etc/passwd`. The following table explains some of the switches for `cut`

Switch	Purpose
`-c`*`RANGE`*	cut out the characters in *`RANGE`*
`-d`*`character`*	specify that the field delimiter is *`character`*
`-f`*`RANGE`*	cut out the fields in *`RANGE`*

Table 3.5
Switches for the `cut` command

`RANGE` used by the `-f` and `-c` switches can take the following forms

- *`number-`*
  get all from character or field *`number`* to the end of the line
- *`number-number2`*
  get all from character or field *`number`* to character or field *`number2`*
- *`number,number2`*
  get characters or fields *`number`* and *`number2`*

And combinations of the above.

Examples

- `cut -c1 /etc/passwd`
  get the first character from every line
- `cut -c1,5,10-20 /etc/passwd`
  get the first, fifth character and every character between 10 and 20

- `cut -d: -f2 /etc/passwd`
  get the second field
- `cut -d: -f3- /etc/passwd`
  get all fields from the third on

paste

This command performs the opposite task to `cut`. It puts lines back together. Assume we have two files

```
names
george
fred
david
janet

addresses
55 Aim avenue
1005 Marks road
5 Thompson Street
43 Pedwell road
```

To put them back together we'd use the command

```
bash$ paste names addresses
george  55 Aim avenue
fred    1005 Marks road
david   5 Thompson Street
janet   43 Pedwell road
```

The two fields have been separated by a tab character. To use a different character you use the `-d switch`.

```
bash$ paste -d: names addresses
george:55 Aim avenue
fred:1005 Marks road
david:5 Thompson Street
janet:43 Pedwell road
```

To paste together lines from the same file you use the `-s switch`.

```
bash$  paste -s names
george  fred    david   janet
```

grep

`grep` stands for **G**lobal **R**egular **E**xpression **P**attern match. It is used to search a file for a particular pattern of characters.

- `grep david /etc/passwd`
  display any line from /etc/passwd that contains david

To get the real power out of `grep you` need to be familiar with regular expressions which are discussed in more detail in a later chapter.

wc

Used to count the number of characters, words and lines in a file. By default it displays all three. Using the switches `-c -w -l` will display the number of characters, words and lines respectively.

```
bash$ wc /etc/passwd
  19      20     697 /etc/passwd
bash$ wc -c /etc/passwd
   697 /etc/passwd
bash$ wc -w /etc/passwd
   20 /etc/passwd
bash$ wc -l /etc/passwd
   19 /etc/passwd
```

For the following exercises create a file called `phone.book` that contains the following

```
george!2334234!55 Aim avenue
fred!343423!1005 Marks road
david!5838434!5 Thompson Street
janet!33343!43 Pedwell road
```

The field delimiter for this file is `!` and the fields are name, phone number, address.

Exercises

3.8. What command would you use to (assume you start from the original file for every question)

1. sort the file on the names
2. sort the file in descending order on phone number
3. display just the addresses
4. change all the `! characters to :`
5. display the first line from the file
6. display the line containing david's information
7. What would effect would the following command have

```
paste -d: -s phone.book
```

3.9. A University student database system must produce a number of files containing information about students in a class. One such example CSV file with 6 fields: student number, surname, firstname, grade (F,P,C,D,HD), mark and degree code is available from the 85321 website/CD-ROM at the URL http://infocom.cqu.edu.au/85321/Resources/Lectures/6/results.csv

Using previous descriptions in the lecture, the 85321 text book and the Linux manual pages come up with commands to perform the following tasks on this example file

1. count the number of students in the class (hint: there is one student per line in the file)
2. display the data file one line at a time
3. get a list of all the student numbers in the class
4. Find all students who received HDs

Getting more out of filters

The filters are a prime example of good UNIX commands. They do one job well and are designed to be chained together. To get the most out of filters you combine them together in long chains of commands. How this is achieved will be examined in a later chapter when the concept of I/O redirection is introduced.

I/O redirection allows you to count the number of people on your computer who have usernames starting with `d` by using the grep command to find all the lines in the /etc/passwd file that start with d and pass the output of that command to the wc command to count the number of matching lines that grep found.

How you do this will be explained next week.

Conclusions

In this chapter you have been provided a brief introduction to the philosophy and format of UNIX commands. In addition some simple commands have been introduced including

- account related commands
  `login passwd exit`
- file and directory manipulation commands
  `cd ls rm mv mkdir`
- some basic commands
  `date who banner cal`
- some filters
  `cat more less head tail sort uniq cut paste tr grep`

A Systems Administrator has to be a "guru". An expert user of their system. A Systems Administrator should not only to be able to get the most out of the system but also to be able to explain and assist other users.

Chapter 4

The File Hierarchy

Introduction

Why?

Like all good operating systems, UNIX allows you the privilege of storing information indefinitely (or at least until the next disk crash) in abstract data containers called files. The organisation, placement and usage of these files comes under the general umbrella of the file hierarchy. As a system administrator, you will need to be very familiar with the file hierarchy. You will use it on a day to day basis as you maintain the system, install software and manage user accounts.

At a first glance, the file hierarchy structure of a typical Linux host (we will use Linux for the basis of our discussion) may appear to have been devised by a demented genius who'd been remiss with their medication. Why, for example, does the root directory contain something like:

```
bin         etc         lost+found  root        usr
boot        home        mnt         sbin        var
dev         lib         proc        tmp
```

Why was it done like this?

Historically, the location of certain files and utilities has not always been standard (or fixed). This has lead to problems with development and upgrading between different "distributions" of Linux *[Linux is distributed from many sources, two major sources are the Slackware and Red Hat package sets]*. The Linux directory structure (or file hierarchy) was based on existing flavours of UNIX, but as it evolved, certain inconsistencies developed. These were often small things like the location (or placement) of certain configuration files, but it resulted in difficulties porting software from host to host.

To combat this, a file standard was developed. This is an evolving process, to date resulting in a fairly static model for the Linux file hierarchy. In this chapter, we will examine how the Linux file hierarchy is structured, how each component relates to the overall OS and why certain files are placed in certain locations.

Linux File System Standard

The location and purposes of files and directories on a Linux machine are defined by the Linux File Hierarchy Standard. The offical website for the Linux File Hierarchy Standard is http://www.pathname.com/fhs/

The important sections

The root of the problem

The top level of the Linux file hierarchy is referred to as the root (or /). The root directory typically contains several other directories including:

Directory	Contains
bin/	Required Boot-time binaries
boot/	Boot configuration files for the OS loader and kernel image
dev/	Device files
etc/	System configuration files and scripts
home/	User/Sub branch directories
lib/	Main OS shared libraries and kernel modules
Lost+found/	Storage directory for "recovered" files
mnt/	Temporary point to connect devices to
proc/	Pseudo directory structure containing information about the kernel, currently running processes and resource allocation
root/	Linux (non-standard) home directory for the root user. Alternate location being the / directory itself
sbin/	System administration binaries and tools
tmp/	Location of temporary files r
usr/	Difficult to define - it contains almost everything else including local binaries, libraries, applications and packages (including X Windows)
var/	Variable data, usually machine specific. Includes spool directories for mail and news

Table 4.1
Major Directories

Generally, the root should not contain any additional files - it is considered bad form to create other directories off the root, nor should any other files be placed there.

Why `root`?

The name "root" is based on the analogous relationship between the UNIX files system structure and a tree! Quite simply, the file hierarchy is an inverted tree.

> I can personally never visiualise an upside down tree – what this phrase really means is that the "top" of the file heirarchy is at one point, like the root of a tree, the bottom is spread out, like the branches of a tree. This is probably a silly analogy because if you turn a tree upside down, you have lots of spreading roots, dirt and several thousand very unhappy worms!

Every part of the file system eventually can be traced back to one central point, the root. The concept of a "root" structure has now been (partially) adopted by other operating systems such as Windows NT. However, unlike other operatings systems, UNIX doesn't have any concept of "drives". While this will be explained in detail in a later chapter, it is important to be aware of the following:

The file system may be spread over several physical devices; different parts of the file heirarchy may exist on totally separate partitions, hard disks, CD-ROMs, network file system shares, floppy disks and other devices.

This separation is transparent to the file system heirarchy, user and applications.

Different "parts" of the file system will be "connected" (or mounted) at startup; other parts will be dynamically attached as required.

The remainder of this chapter examines some of the more important directory structures in the Linux file hierarchy.

Homes for users

Every user needs a home...

The `/home` directory structure contains the the home directories for most login-enabled users (some notable exceptions being the root user and (on some systems) the www/web user). While most small systems will contain user directories directly off the `/home` directory (for example, `/home/jamiesob`), on larger systems is common to subdivide the home structure based on classes (or groups) of users, for example:

```
/home/admin             # Administrators
/home/finance           # Finance users
/home/humanres          # Human Resource users
/home/mgr               # Managers
/home/staff             # Other people
```

Other homes?

`/root` is the home directory for the root user. If, for some strange reason, the `/root` directory doesn't exist, then the root user will be logged in in the `/` directory - this is actually the traditional location for root users.

There is some debate as to allowing the root user to have a special directory as their login point - this idea encourages the root user to set up their `.profile`, use "user" programs like `elm`, `tin` and `netscape` (programs which require a home directory in which to place certain configuration files) and generally use the root account as a beefed up user account. A system administrator should **never** use the root account for day to day user-type interaction; the root account should only be used for system administration purposes only.

> Be aware that you must be extremely careful when allowing a user to have a home directory in a location other than the `/home` branch. The problem occurs when you, as a system administrator, have to back-up the system - it is easy to miss a home directory if it isn't grouped with others in a common branch (like `/home`).

`/usr` and `/var`

And the difference is...

It is often slightly confusing to see that `/usr` and `/var` both contain similar directories:

```
/usr
X11R6             games             libexec           src
bin               i486-linux-libc5  local             tmp
dict              include           man
doc               info              sbin
etc               lib               share

/var
catman    local     log       preserve  spool
lib       lock      nis       run       tmp
```

It becomes even more confusing when you start examining the the maze of links which intermingle the two major branches.

> Links are a way of referencing a file or directory by many names and many locations within the file hierarchy. They are effectively like "pointers" to files - think of them as like leaving a post-it note saying "see this file". Links will be explained in greater detail in the next chapter.

To put it simply, `/var` is for **VAR**iable data/files. `/usr` is for **US**e**R** accessible data, programs and libraries. Unfortunately, history has confused things - files which should have been placed in the `/usr` branch have been located in the

/var branch and vice versa. Thus to "correct" things, a series of links have been put in place. Why the reason for the separation? Does it matter. The answer is: Yes, but No :)

Yes in the sense that the file standard dictates that the /usr branch should be able to be mounted (another way of saying "attached" to the file hierarchy - this will be covered in the next chapter) **READ ONLY** (thus can't contain variable data). The reasons for this are historical and came about because of something called NFS exporting.

> NFS exporting is the process of one machine (a server) "exporting" its copy of the /usr structure (and others) to the network for other systems to use.

If several systems were "sharing" the same /usr structure, it would not be a good idea for them all to be writing logs and variable data to the same area! It is also used because minimal installations of Linux can use the /usr branch directly from the CDROM (a read-only device).

However, it is "No" in the sense that:

- /usr is usually mounted READ-**WRITE**-EXECUTE on Linux systems anyway
- In the author's experience, exporting /usr READ-ONLY via NFS isn't entirely successful without making some very non-standard modifications to the file hierarchy!

The following are a few highlights of the /var and /usr directory branches:

/usr/local

All software that is installed on a system after the operating system package itself should be placed in the /usr/local directory. Binary files should be located in the /usr/local/bin (generally /usr/local/bin should be included in a user's PATH setting). By placing all installed software in this branch, it makes backups and upgrades of the system far easier - the system administrator can back-up and restore the entire /usr/local system with more ease than backing-up and restoring software packages from multiple branches (i.e.. /usr/src, /usr/bin etc.).
An example of a /usr/local directory is listed below:

```
bin       games         lib           rsynth           cern
man       sbin          volume-1.11   info
mpeg      speak         www           etc              java
netscape  src
```

As you can see, there are a few standard directories (bin, lib and src) as well as some that contain installed programs.

lib, include and src

Linux is a very popular platform for C/C++, Java and Perl program development. As we will discuss in later chapters, Linux also allows the system administrator to actually modify and recompile the kernel. Because of this, compilers, libraries and source directories are treated as "core" elements of the file hierarchy structure.

The /usr structure plays host to three important directories:

/usr/include holds most of the standard C/C++ header files - this directory will be referred to as the primary include directory in most Makefiles.

> Makefiles are special script-like files that are processed by the make program for the purposes of compiling, linking and building programs.

/usr/lib holds most static libraries as well as hosting subdirectories containing libraries for other (non C/C++) languages including Perl and TCL. It also plays host to configuration information for ldconfig.

/usr/src holds the source files for most packages installed on the system. This is traditionally the location for the Linux source directory (/usr/src/linux), for example:

```
linux          linux-2.0.31  redhat
```

> Unlike DOS/Windows based systems, most Linux programs usually come as source and are compiled and installed locally

/var/spool

This directory has the potential for causing a system administrator a bit of trouble as it is used to store (possibly) large volumes of temporary files associated with printing, mail and news. /var/spool may contain something like:

```
at          lp          lpd         mqueue      samba       uucppublic
cron        mail        rwho        uucp
```

In this case, there is a printer spool directory called lp (used for storing print request for the printer lp) and a /var/spool/mail directory that contains files for each user's incoming mail.

> Keep an eye on the space consumed by the files and directories found in /var/spool. If a device (like the printer) isn't working or a large volume of e-mail has been sent to the system, then much of the hard drive space can be quickly consumed by files stored in this location.

X Windows

X-Windows provides UNIX with a very flexible graphical user interface. Tracing the X Windows file hierarchy can be very tedious, especially when your are trying to locate a particular configuration file or trying to removed a stale lock file.

> A lock file is used to stop more than one instance of a program executing at once, a stale lock is a lock file that was not removed when a program terminated, thus stopping the same program from restarting again

Most of X Windows is located in the `/usr` structure, with some references made to it in the `/var` structure.

Typically, most of the action is in the `/usr/X11R6` directory (this is usually an alias or link to another directory depending on the release of X11 - the X Windows manager). This will contain:

```
bin      doc    include  lib      man
```

The main X Windows binaries are located in `/usr/X11R6/bin`. This may be accessed via an alias of `/usr/bin/X11` .

Configuration files for X Windows are located in `/usr/X11R6/lib`. To really confuse things, the X Windows configuration utility, `xf86config`, is located in `/usr/X11R6/bin`, while the configuration file it produces is located in `/etc/X11` (`XF86Config`)!

Because of this, it is often very difficult to get an "overall picture" of how X Windows is working - my best advice is read up on it before you start modifying (or developing with) it.

Bins

Which bin?

A very common mistake amongst first time UNIX users is to incorrectly assume that all "bin" directories contain temporary files or files marked for deletion. This misunderstanding comes about because:

- People associate the word "bin" with rubbish
- Some unfortunate GUI based operating systems use little icons of "trash cans" for the purposes of storing deleted/temporary files.

However, bin is short for binary - binary or executable files. There are four major bin directories (none of which should be used for storing junk files :)

- `/bin`
- `/sbin`
- `/usr/bin`
- `/usr/local/bin`

Why so many?

All of the bin directories serve similar but distinct purposes; the division of binary files serves several purposes including ease of backups, administration and logical separation. Note that while most binaries on Linux systems are found in one of these four directories, not all are.

/bin

This directory must be present for the OS to boot. It contains utilities used during the startup; a typical listing would look something like:

```
 Mail           df             gzip           mount          stty
 arch           dialog         head           mt             su
 ash            dircolors      hostname       mt-GNU         sync
 bash           dmesg          ipmask         mv             tar
 cat            dnsdomainname  kill           netstat        tcsh
 chgrp          domainname     killall        ping           telnet
 chmod

domainname-yp  ln             ps             touch
 chown          du             login          pwd            true
 compress       echo           ls             red            ttysnoops
 cp             ed             mail           rm             umount
 cpio

false          mailx          rmdir          umssync
 csh            free           mkdir          setserial      uname
 cut            ftp            mkfifo         setterm        zcat
 date           getoptprog     mknod          sh             zsh
 dd             gunzip         more           sln
```

Note that this directory contains the shells and some basic file and text utilities (`ls, pwd, cut, head, tail, ed` etc). Ideally, the `/bin` directory will contain as few files as possible as this makes it easier to take a direct copy for recovery boot/root disks.

/sbin

`/sbin` Literally "System Binaries". This directory contains files that should generally only be used by the root user, though the Linux file standard dictates that no access restrictions should be placed on normal users to these files. It should be noted that the `PATH` setting for the root user includes `/sbin`, while it is (by default) not included in the `PATH` of normal users.

The `/sbin` directory should contain essential system administration scripts and programs, including those concerned with user management, disk administration, system event control (restart and shutdown programs) and certain networking programs.

As a general rule, if users need to run a program, then it should not be located in `/sbin`. A typical directory listing of `/sbin` looks like:

```
  adduser          ifconfig         mkfs.minix       rmmod
  agetty           init             mklost+found     rmt
  arp              insmod           mkswap           rootflags
  badblocks        installpkg       mkxfs            route
  bdflush          kbdrate          modprobe         runlevel
  chattr           killall5         mount            setup
  clock            ksyms            netconfig        setup.tty
  debugfs          ldconfig         netconfig.color  shutdown
  depmod           lilo             netconfig.tty    swapdev
  dosfsck          liloconfig       pidof            swapoff
  dumpe2fs         liloconfig-color pkgtool          swapon
  e2fsck           lsattr           pkgtool.tty      telinit
  explodepkg       lsmod            plipconfig       tune2fs
  fdisk            makebootdisk     ramsize          umount
```

```
fsck             makepkg          rarp             update
 fsck.minix        mkdosfs           rdev              vidmode
 genksyms          mke2fs            reboot            xfsck
 halt              mkfs             removepkg
```

> The very important `ldconfig` program is also located in `/sbin`. While not commonly used from the shell prompt, `ldconfig` is an essential program for the management of dynamic libraries (it is usually executed at boot time). It will often have to be manually run after library (and system) upgrades.

You should also be aware of:
`/usr/sbin` - used for non-essential admin tools.
`/usr/local/sbin` - locally installed admin tools.

`/usr/bin`

This directory contains most of the user binaries - in other words, programs that users will run. It includes standard user applications including editors and email clients as well as compilers, games and various network applications.

A listing of this directory will contain some 400 odd files. Users should definitely have `/usr/bin` in their `PATH` setting.

`/usr/local/bin`

To this point, we have examined directories that contain programs that are (in general) part of the actual operating system package. Programs that are installed by the system administrator after that point should be placed in `/usr/local/bin`. The main reason for doing this is to make it easier to back up installed programs during a system upgrade, or in the worst case, to restore a system after a crash.

> The `/usr/local/bin` directory should only contain binaries and scripts - it should not contain subdirectories or configuration files.

Configuration files, logs and other bits!

etc etc etc.

`/etc` is one place where the root user will spend a lot of time. It is not only the home to the all important `passwd` file, but contains just about every configuration file for a system (including those for networking, X Windows and the file system).

The `/etc` branch also contains the `skel`, `X11` and `rc.d` directories.

`/etc/skel` contains the skeleton user files that are placed in a user's directory when their account is created.

`/etc/X11` contains configuration files for X Windows.

`/etc/rc.d` is contains rc directories - each directory is given by the name `rcn.d` (`n` is the run level) - each directory may contain multiple files that will be executed at the particular run level. A sample listing of a `/etc/rc.d` directory looks something like:

```
init.d      rc.local    rc0.d       rc2.d       rc4.d       rc6.d
rc          rc.sysinit  rc1.d       rc3.d       rc5.d
```

Logs

Linux maintains a particular area in which to place logs (or files which contain records of events). This directory is `/var/log`.

This directory usually contains:

```
cron          lastlog      maillog.2    samba-log.   secure.2     uucp
cron.1        log.nmb      messages     samba.1      sendmail.st  wtmp
cron.2        log.smb      messages.1   samba.2      spooler      xferlog
dmesg         maillog      messages.2   secure       spooler.1    xferlog.1
httpd         maillog.1    samba        secure.1     spooler.2    xferlog.2
```

`/proc`

The `/proc` directory hierarchy contains files associated with the ***executing*** kernel. The files contained in this structure contain information about the state of the system's resource usage (how much memory, swap space and CPU is being used), information about each process and various other useful pieces of information. We will examine this directory structure in more depth in later chapters.

> The `/proc` file system is the main source of information for a program called **`top`**. This is a very useful administration tool as it displays a "live" readout of the CPU and memory resources being used by each process on the system.

`/dev`

We will be discussing `/dev` in detail in the next chapter, however, for the time being, you should be aware that this directory is the primary location for special files called **device files**.

Conclusion

Future standards

Because Linux is a dynamic OS, there will no doubt be changes to its file system as well. Two current issues that face Linux are:

- Porting Linux on to may architectures and requiring a common location for hardware independent data files and scripts - the current location is /usr/share - this may change.
- The location of third-party commercial software on Linux systems - as Linux's popularity increases, more software developers will produce commercial software to install on Linux systems. For this to happen, a

location in which this can be installed must be provided and enforced within the file system standard. Currently, `/opt` is the likely option.

Because of this, it is advisable to obtain and read the latest copy of the file system standard so as to be aware of the current issues. Other information sources are easily obtainable by searching the web.

You should also be aware that while (in general), the UNIX file hierarchy looks similar from version to version, it contains differences based on requirements and the history of the development of the operating system implementation.

Review Questions

4.1

You have just discovered that the previous system administrator of the system you now manage installed netscap in /sbin. Is this an appropiate location? Why/Why not?.

4.2

Where are man pages kept? Explain the format of the man page directories. (Hint: I didn't explain this anywhere in this chapter - you may have to do some looking)

4.3

As a system administrator, you are going to install the following programs, in each case, state the likely location of each package:

- Java compiler and libraries
- DOOM (a loud, violent but extremely entertaining game)
- A network sniffer (for use by the sys admin only)
- A new kernel source
- A X Windows manager binary specially optimised for your new monitor

Chapter 5
Processes and Files

Introduction

This chapter introduces the important and related UNIX concepts of processes and files.

A process is basically an executing program. All the work performed by a UNIX system is carried out by processes. The UNIX operating system stores a great deal of information about processes and provides a number of mechanisms by which you can manipulate both the files and the information about them.

All the long term information stored on a UNIX system, like most computers today, is stored in files which are organised into a hierarchical directory structure. Each file on a UNIX system has a number of attributes that serve different purposes. As with processes there are a collection of commands which allow users and Systems Administrators to modify these attributes.

Among the most important attributes of files and processes examined in this chapter are those associated with user identification and access control. Since UNIX is a multi-user operating system it must provide mechanisms which restrict what and where users (and their processes) can go. An understanding of how this is achieved is essential for a Systems Administrator.

Other Resources

Other resources which discuss some of the concepts mentioned in this chapter include

- Chapter 17 of this text
  This is the security chapter of the text and not surprisingly it includes a discussion of file permissions including some additional material which is not discussed here. This chapter is actually a copy of the Security HOW-TO from the LDP.
- Online lecture 5 (which includes slides and audio)
  Included on the 85321 website/CD-ROM this lecture discusses many of the topics covered in this chapter. You may find it useful to take a listen to this lecture as a supplement to the chapter.
- Guides on the LDP
  The Linux Installation and Getting Started Guide has a number of sections looking at the permissions and job control

Multiple users

UNIX is a multi-user operating system. This means that at any one time there are multiple people all sharing the computer and its resources. The operating system must have some way of identifying the users and protecting one user's resources from the other users.

Identifying users

Before you can use a UNIX computer you must first log in. The login process requires that you have a username and a password. By entering your username you identify yourself to the operating system.

Users and groups

In addition to a unique username UNIX also places every user into at least one group. Groups are used to provide or restrict access to a collection of users and are specified by the `/etc/group` file.

To find out what groups you are a member of use the `groups` command. It is possible to be a member of more than one group.

The following example is taken from my Redhat 6.1 machine

```
[david@faile links]$ groups
david
```

Executing the groups command as the "normal" user david shows that he is only a member of the david group. Under Linux when you create a user with the adduser command the default action is to create a group with the same name as the account.

In the following I user the su command to change to the root user (this requires that I enter root's password). Remember you should do the absolute minimum as root.

```
[david@faile links]$ su -
Password:
[root@faile /root]# groups
root bin daemon sys adm disk wheel
```

From this you can see that the root user is a member of a number of groups.

Names and numbers

As you've seen each user and group has a unique name. However the operating system does not use these names internally. The names are used for the benefit of the human users.

For its own purposes the operating system actually uses numbers to represent each user and group (numbers are more efficient to store). This is achieved by each username having an equivalent user identifier (UID) and every group name having an equivalent group identifier (GID).

The association between username and UID is stored in the `/etc/passwd` file. The association between group name and GID is stored in the `/etc/group` file.

To find out the your UID and initial GID try the following command

```
grep username /etc/passwd
```

Where *username* is your username. This command will display your entry in the `/etc/passwd` file. The third field is your UID and the fourth is your initial GID. On my system my UID is 500 and my GID is 100.

```
bash$  grep david /etc/passwd
david:*:500:100:David Jones:/home/david:/bin/bash
```

id

The `id` command can be used to discover username, UID, group name and GID of any user.

```
dinbig:~$ id
uid=500(david) gid=100(users) groups=100(users)
dinbig:~$ id root
uid=0(root) gid=0(root) groups=0(root),1(bin),
2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy)
```

In the above you will see that the user `root` is a member of more than one group. The entry in the `/etc/passwd` file stores the GID of the users initial group (mine is 100, root's is 0). If a user belongs to any other groups they are specified in the `/etc/group` file.

Commands and processes

Whenever you run a program, whether it is by typing in at the command line or running it from X-Windows, a process is created. It is the process, a program in execution and a collection of executable code, data and operating system data structures, which perform the work of the program.

The UNIX command line that you use to enter commands is actually another program/command called the shell. The shell is responsible for asking you for a command and then attempting to execute the command. (The shell also performs a number of other tasks which are discussed in the next chapter).

Where are the commands?

For you to execute a command, for example `ls`, that command must be in one of the directories in your search path. The search path is a list of directories maintained by the shell.

When you ask the shell to execute a command it will look in each of the directories in your search path for a file with the same name as the command. When it finds the executable program it will run it. If it doesn't find the executable program it will report *command_name*`: not found`.

which

Linux and most UNIX operating systems supply a command called `which`. The purpose of this command is to search through your search path for a particular command and tell you where it is.

For example, the command `which ls` on my machine `aldur` returns `/usr/bin/ls`. This means that the program for `ls` is in the directory `/usr/bin`. If you do which for ls on a Redhat Linux machine you will get a different location.

Exercises

5.1.	Use the `which` command to find the locations of the following commands
	`ls`
	`echo`
	`set`

Why can't I run my shell script?

When you get to chapter 8 of the textbook you will be introduced to shell scripts. Shell scripts are small executable files that contain a bunch of commands, somewhat like batch files under MS-DOS (only better). A common problem many people have when they create their first shell script is that it can't be found.

For example, let's assume I've create a shell script called hello in the current directory. The problem goes something like this.

```
[david@faile links]$ pwd
/home/david/teaching/85321/2000/textbook/mine/links
[david@faile links]$ ls -l hello
-rwxrwxr-x   1 david    david          34 Jan  8 17:15 hello
[david@faile links]$ hello
bash: hello: command not found
```

To start with I find out what the current directory is, you will see why in the next couple of paragraphs. I then use the `ls` command to confirm that the executable file hello is located in the current directory. Then, at last, I've tried to execute it but get an error message. As mentioned above "command not found" means that the shell was unable to locate the executable file in the current search path.

If you think about it you should figure out that this means that the current directory is not in the search path. That's why the shell can't find the command hello.

There are two solutions to this problem

Tell the shell exactly the location of the `hello` executable file.
By just typing the name of the command I am telling the shell to search the path. I can be a little more specific with the location using either relative or absolute paths.

```
[david@faile links]$
/home/david/teaching/85321/2000/textbook/mine/links/hello
hello david, how are you
[david@faile links]$ ./hello
hello david, how are you
```

Include the current directory in the search path.
The idea is to modify the search path so that the shell also looks in the current

directory. Absolute and relative paths play a part here also. You will see an explanation of how to change the path in a later chapter.

```
[david@faile links]$ PATH=$PATH:.
[david@faile links]$ hello
hello david, how are you
```

When is a command not a command?

In the previous exercise you will have discovered that `which` could not find the `set` command. How can this be possible? If I enter the `set` command on my Linux box it works fine. So if all commands are executable files in the search path then why can't `which` find it?

This is because `set` is a built-in shell command. This means there isn't an executable program that contains the code for the `set` command. Instead the code for `set` is actually built into the shell. In other words no matter how hard you look you won't find an executable file called set.

So, as mentioned before any command you execute at a UNIX command line falls into one of two categories

A shell command.
This is a command which is understood by the shell you are using. It isn't an executable file.

An executable file.
The executable file will be located somewhere in your search path. When you execute this type of command the shell will search for the file and then create a process which executes this file.

Why shell commands are faster than other commands

As mentioned above executing a shell command does not require the creation of a new process, the existing shell process executes the command. For normal commands a new process must be created.

Creating a new process is, relatively speaking, quite a long process. This is especially true when the executable file must be read from disk (you should remember from operating systems that reading from disk is very, very slow when compared to RAM and CPU operations).

This is why internal shell commands are much faster than normal commands.

For example, I have created two shell scripts (add and add2) which both perform the simple task of adding up to 1000 1 at a time. add uses a normal command to perform the addition. While add2 uses an internal shell command to perform the addition. To compare the speed of the two scripts I use the UNIX time command to work out how long each script takes to execute

```
[david@faile links]$ time add
6.82user 7.15system 0:13.97elapsed 99%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (107194major+70036minor)pagefaults 0swaps
[david@faile links]$ time add2
0.52user 0.00system 0:00.51elapsed 100%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (194major+24minor)pagefaults 0swaps
```

From the output of these two commands you should be able to see that using the internal shell command is significantly faster than using the normal UNIX command. The drawback of shell commands is that they can only be used with a specific shell, you might not be using the right shell. On the other hand, the common UNIX commands are present on all UNIX systems.

Controlling processes

Processes are the main active component of any modern operating system. Any work performed by a modern operating system is performed by processes. UNIX/Linux is no different. This section provides an overview of how you can view and manipulate processes as a normal user. This is a primary responsibility for a Systems Administrator so it is important foundation knowledge.

In this section you will learn how to

- view existing processes
  Discover how to find out which processes exist, what is their current state and who they belong to.
- Job control
  How you can control the execution of processes using the features of common shells.
- Process manipulation
  How processes can be stopped or restarted by sending signals.

Online lecture 5 also takes a look at this material.

Viewing existing processes

As mentioned earlier in this chapter every UNIX command you execute runs as a new process. Since Linux/UNIX is a multi-tasking operating system at any one time there can be tens, hundreds even thousands of processes running (the limit is set by a value in the source code for the Linux kernel).

As a Systems Administrator and a normal user you will want to be able to find out which processes are currently running, what there current state is and a bunch of other process related information. This section introduces you to a number of commands which allow you to do this including

- `ps`
  Provides a snapshot of the processes which are currently running.
- `top`
  Provides a full screen, updated view of the current processes.
- `pstree`
  Displays a tree-like structure of the current processes.
- Various graphical tools
  It is now common for a range of GUI tools to be available. This section will look briefly at those which come with the GNOME desktop environment.

ps

Any use on a UNIX system can execute the ps command and see something like

```
[david@faile linux]$ ps
  PID TTY          TIME CMD
  667 pts/0    00:00:00 bash
  893 pts/0    00:00:00 ps
```

This is simply the list of all processes running from the current terminal (TTY which is currently pts/0). The ps command understands a wide range of command-line switches which will modify both the

- rows, and
  By modifiying the rows which appear you are changing which processes are shown. By default you are only seeing the processes for the current terminal. The example below shows how this can be changed.
- columns
  The columns display various bits of information about the processes. By default you see such things as the commands used the process is running (the CMD column) and the unique process identifier for the process (the PID column).

For example,

```
[david@faile linux]$ ps a
  PID TTY      STAT   TIME COMMAND
  667 pts/0    S      0:00 bash
  902 pts/0    R      0:00 ps a
[david@faile linux]$ ps x
  PID TTY      STAT   TIME COMMAND
  592 tty1     SW     0:00 [bash]
  603 tty1     SW     0:00 [startx]
  610 tty1     SW     0:00 [xinit]
  615 tty1     S      0:00 /usr/bin/gnome-session

           ..... some output deleted here...

  667 pts/0    S      0:00 bash
  669 tty1     SW     0:00 [gnome-pty-helpe]
  670 pts/1    SW     0:00 [bash]
  671 tty1     SW     0:00 [gnome-pty-helpe]
  672 pts/2    SW     0:00 [bash]
  675 tty1     SW     0:00 [gnome-pty-helpe]
  676 tty1     SW     0:00 [gnome-pty-helpe]
  677 tty1     SW     0:00 [gnome-pty-helpe]
  678 pts/3    S      0:00 bash
  679 pts/4    S      0:00 bash
  680 pts/5    SW     0:00 [bash]
  688 tty1     S      1:42 /home/david/Office51/bin/soffice.bin
  707 tty1     S      0:41 /usr/lib/netscape/netscape-communicator -irix-
session
  720 tty1     S      0:00 (dns helper)
  721 tty1     S      0:00 /home/david/Office51/bin/soffice.bin
  722 tty1     S      0:00 /home/david/Office51/bin/soffice.bin
  723 tty1     S      0:00 /home/david/Office51/bin/soffice.bin
  724 tty1     S      0:00 /home/david/Office51/bin/soffice.bin
  725 tty1     S      0:00 /home/david/Office51/bin/soffice.bin
  727 tty1     S      0:00 /home/david/Office51/bin/soffice.bin
  795 pts/3    S      0:00 vi TODO
  835 tty1     S      0:26 gtop
  924 pts/0    R      0:00 ps x
```

Refer to the manual page for the ps command for more information about the available switches. You will notice that ps does not follow the standard UNIX command format. In this case the command-line switches a and x were not preceded with -.

5.2. Use the ps command to discover which user owns the
```
/usr/sbin/atd
sendmail
```
processes.

top

ps provides a one-off snapshot of the current processes. If you want an on-going view of the processes you need to use top. top produces output something like

```
 10:56am  up  1:21,  7 users,  load average: 1.32, 0.80, 0.41
95 processes: 92 sleeping, 2 running, 1 zombie, 0 stopped
CPU states: 15.0% user,  3.6% system,  0.0% nice, 81.3% idle
Mem:  127948K av, 124496K used,   3452K free,  58884K shrd,   2888K buff
Swap:  72252K av,  21956K used,  50296K free                 34528K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COMMAND
  974 david     12   0   428  428   348 R       0 79.0  0.3   4:00 yes
  977 david      7   0  1044 1044   816 R       0 10.6  0.8   0:00 top
  835 david      1   0  3912 3912  2876 S       0  9.7  3.0   1:55 gtop
  611 root       0   0 34468  32M  1620 S       0  0.8 25.8   3:00 X
    1 root       0   0   124   72    52 S       0  0.0  0.0   0:04 init
    2 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kflushd
    3 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kupdate
    4 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kpiod
    5 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kswapd
    6 root     -20 -20     0    0     0 SW<     0  0.0  0.0   0:00
mdrecoveryd
  253 bin        0   0    80    0     0 SW      0  0.0  0.0   0:00 portmap
  269 root       0   0   380  368   324 S       0  0.0  0.2   0:00 apmd
  322 root       0   0   200  148   104 S       0  0.0  0.1   0:00 syslogd
  333 root       0   0   504  160   112 S       0  0.0  0.1   0:00 klogd
  349 daemon     0   0   144  104    76 S       0  0.0  0.0   0:00 atd
  365 root       0   0   240  188   144 S       0  0.0  0.1   0:00 crond
  380 root       0   0   140    0     0 SW      0  0.0  0.0   0:00 cardmgr
```

As with ps there are a number of command-line switches which modify the operation of top. Additionally top has a number of interactive commands you can use while it is running. For example, hitting the h key while top is running will display a simple help screen which lists the interactive commands.

pstree and ps f

Each new process (the child process) must be started by another process (the parent process). As a result UNIX processes form a family tree. The pstree and the f switch of the ps command allow you to view this family tree. For example

```
[david@faile david]$ pstree
init-+-apmd
     |-atd
     |-cardmgr
     |-crond
     |-enlightenment
     |-gen_util_applet
```

```
|-gmc
|-gnome-name-serv
|-gnome-smproxy
|-gnome-terminal-+-bash---pstree
|                `-gnome-pty-helpe
|-3*[gnome-terminal-+-bash]
|                   `-gnome-pty-helpe]
|-gnome-terminal-+-bash-+-top
|                |      `-yes
|                `-gnome-pty-helpe
|-gnome-terminal-+-bash---su---bash
|                `-gnome-pty-helpe
|-gnomepager_appl
|-gpm
|-gtop
|-httpd---15*[httpd]
|-inetd
|-kflushd
|-klogd
|-kpiod
|-kswapd
|-kupdate
|-login---bash---startx---xinit-+-X
|                               `-gnome-session
|-lpd
|-magicdev
|-mdrecoveryd
|-5*[mingetty]
|-msql2d
|-netscape-commun---2*[netscape-commun]
|-panel
|-portmap
|-safe_mysqld---mysqld---mysqld---mysqld
|-soffice.bin---soffice.bin---5*[soffice.bin]
|-syslogd
|-xfs
`-xscreensaver
```

gtop

The increasing use of X Windows and GUI environments means that there have been a number of GUI tools written to provide similar features as the text-based tools introduced in the previous couple of sections. One of them is gtop, the GNU system monitor program, which by default provides a display not unlike top (but as GUI). Gtop also provides a number of additional services including displays of memory and file system usage. Diagram 5.1 is a screen shot of the memory usage screen.

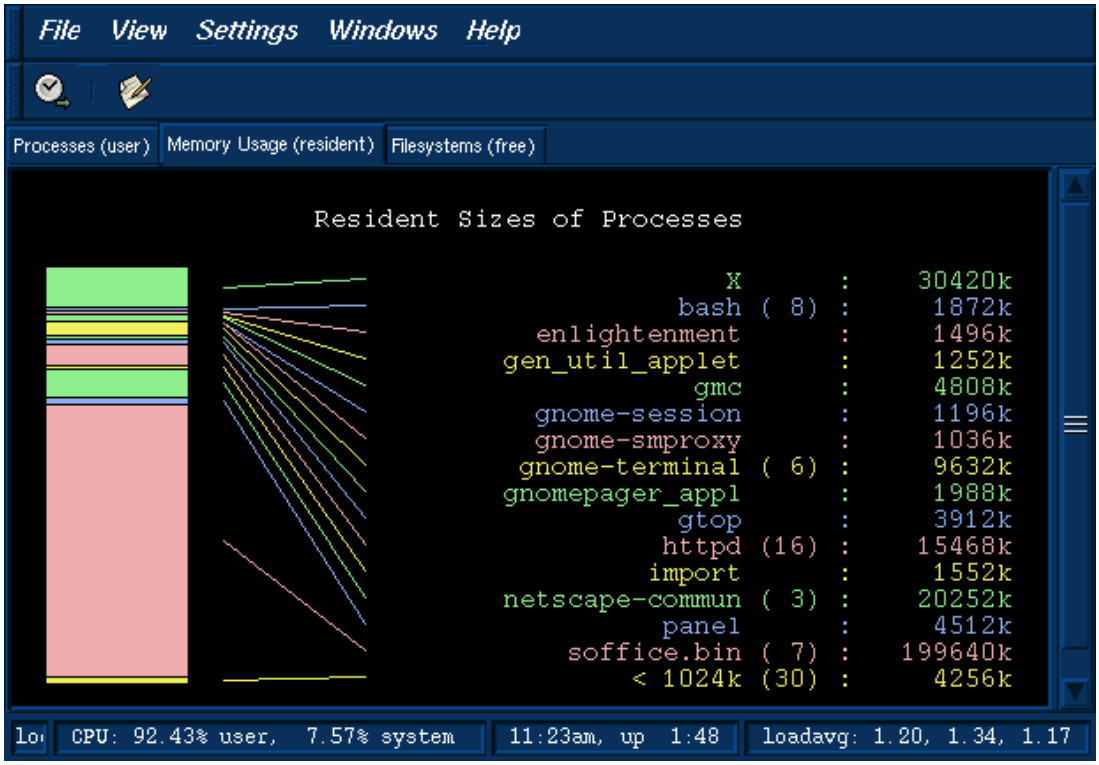


Diagram 1.1
Screen shot of gtop

Job control

Jobs and processes are the same thing in UNIX terminology. Job control is a facility provided by most shells which allow you to switch between multiple running processes.

So far most of you will have been running only a single job. Not unlike what was done in the previous examples when running the ps command. The normal process goes something like this

- You type a command at the shell prompt
- The shell runs that command while you wait for it complete.
- When it is finished the shell displays another command-line and you can start again.

During this process the shell goes "to sleep" waiting for the command to finish. You can see this in the ps a example from above. In this example bash is the shell and ps is the command which is being executed. Take a look at the STAT column for bash, it is S. STAT or status indicates the current status for a process. NewTable 5.1 summarises the possible states for a Linux process. This table is adapted from the manual page for the ps command.

Process	State codes
D	uninterruptible sleep (usually IO)
R	runnable (on run queue)
S	Sleeping
T	traced or stopped
Z	a defunct ("zombie") process

NewTable 5.1
Linux Process States

As you should remember from operating systems on a single CPU system there can only ever be one running process. In the ps a example from above the running process is the one executing the ps command.

This running process is called the foreground process (job). It is the process which "owns" the terminal for input and output. Usually there is only one running process. However most shells provide mechanisms by which you can

- interrupt a process
  Interrupting a process is the same as killing it. The process dies i.e. is no longer running. The typical method for interrupting the current foreground process is using the CTRL-C key combination (hold the control key down and hit the c key).
  For example, run the yes command which continues to display a line of y's one to a line. The yes command will quite happily do this forever. To stop it hit CTRL-C. You have just interrupted a process.

- suspend a process
  Suspending a process puts it to sleep until you start it again. You use the key combination CTRL-Z to suspend a process. Run the yes command again. This time suspend it rather than interrupt it. You should see something like

```
y
  y

  [1]+  Stopped                 yes
```

The [1] is the job number for the suspended process. You can use this to restart the process. If you now run the ps a command you will see something like

```
[david@faile 2000]$ ps a
  PID TTY      STAT   TIME COMMAND
  678 pts/3    S      0:00 bash
  962 pts/3    T      0:00 yes
  963 pts/3    R      0:00 ps a
```

Notice that the yes process appears, so it still exists. If you refer back to the previous table you can see that its state is now stopped.

- check on the status of jobs
  The jobs command is used to check on the status of the jobs you currently

have associated with the terminal. In our current situation you get something like

```
[david@faile 2000]$ jobs
[1]+  Stopped                 yes
```

- change the current foreground process
  To put the yes command back into the foreground (to take it out of the background) you can use the fg command. fg %1 will put the yes command back into the foreground and start the y's scrolling down the screen again. The %1 is used to indicate which job you want back into the foreground. The 1 matches the [1] displayed when we stopped the job above. Feel free to interrupt the job at any stage.

- run other processes in the background
  The shells also support the idea of starting a process off in the background. This means that the command you execute goes straight into the background rather than staying in the foreground. This is achieved using the & symbol. For example

```
[david@faile 2000]$ yes > /dev/null &
[1] 974
[david@faile 2000]$ jobs
[1]+  Running                 yes >/dev/null &
[david@faile 2000]$ ps a
  PID TTY      STAT   TIME COMMAND
  678 pts/3    S      0:00 bash
  974 pts/3    R      0:35 yes
  976 pts/3    R      0:00 ps a
```

The [1] 974 indicates that the yes command has become job number 1 with process id 974. This is reinforced by using the jobs and ps a commands to view the current jobs and processes. Notice that we now have two processes which are on the runnable queue, ps and yes.

Manipulating processes

You have already seen some simple approaches to manipulating processes using the CTRL-C and CTRL-Z key combinations. These approaches along with all approaches to manipulating processes are related to sending signals to processes. When a process is executed it automatically has a collection of signal handlers create. Each signal handler is essentially a function which is executed when a certain signal is received.

If you are interested in finding out more about signals you can refer to online lecture 5 or to the manual page signal(7). This manual page describes all 30 signals used by Linux and also the default actions which are expected as a result of receiving a particular signal.

The `kill` command

Apart from using certain key combinations you can also send signals to processes using the kill command. The kill command is used to send a specific signal to a specific process. This means you usually have to specify both the signal and the process.

By default the kill command sends the TERM signal. You can specify other signals by using the appropriate number of title. The -l switch of the kill command provides a quick overview of the available signals, their names and numbers.

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

You specify the process to which you want to send a signal using the process identifier as shown by the ps or top commands. The following commands demonstrate how job control, the ps command and the kill command can be combined.

```
[david@faile 2000]$ yes > /dev/null &
[2] 1187
[1]   Killed                  yes >/dev/null
[david@faile 2000]$ yes > /dev/null &
[3] 1188
[david@faile 2000]$ yes > /dev/null &
[4] 1189
[david@faile 2000]$ ps a
  PID TTY      STAT   TIME COMMAND
  678 pts/3    S      0:00 bash
 1187 pts/3    R      0:13 yes
 1188 pts/3    R      0:11 yes
 1189 pts/3    R      0:11 yes
 1190 pts/3    R      0:00 ps a
```

To start with we create three versions of the yes command all running in the background. We know start sending some signals to the processes using the kill command.

In the first kill command I don't specify a signal. This means the kill command will use the default TERM signal. The names of signals are shown in the kill -l output from above. However, you won't see a name TERM, you will see the name SIGTERM. When used in the kill command and in some discussions the SIG is dropped. So the KILL signal is called SIGKILL above.

```
[david@faile 2000]$ kill 1187
[david@faile 2000]$ ps a
  PID TTY      STAT   TIME COMMAND
  678 pts/3    S      0:00 bash
 1188 pts/3    R      0:40 yes
 1189 pts/3    R      0:39 yes
 1193 pts/3    R      0:00 ps a
[2]   Terminated              yes >/dev/null
```

From the message and the output of the ps command you can see that process 1187 has been destroyed.

```
[david@faile 2000]$ kill -STOP 1188

[3]+  Stopped (signal)        yes >/dev/null
[david@faile 2000]$ kill -19 1189
```

```
[david@faile 2000]$

[4]+  Stopped (signal)        yes >/dev/null
[david@faile 2000]$ ps a
  PID TTY      STAT   TIME COMMAND
  678 pts/3    S      0:00 bash
 1188 pts/3    T      0:53 yes
 1189 pts/3    T      1:11 yes
 1195 pts/3    R      0:00 ps a
```

In the previous commands the two processes 1188 and 1189 have been suspended using the kill command instead of using the CTRL-Z key combination. This demonstrates that when you use the CTRL-Z key combination you are actually sending the process the SIGSTOP (signal number 19) signal.

```
[david@faile 2000]$ kill -5 1188
[david@faile 2000]$ ps a
  PID TTY      STAT   TIME COMMAND
  678 pts/3    S      0:00 bash
 1188 pts/3    T      0:53 yes
 1189 pts/3    T      1:11 yes
 1200 pts/3    R      0:00 ps a
```

From these commands it appears that sending signal 5 (SIGTRAP) has no effect on process 1188.

Exercises

5.3.	Under the VMS operating system it is common to use the key combination `CTRL-Z` to kill a program. A new user on your UNIX system has been using VMS a lot. What happens when he uses `CTRL-Z` while editing a document with `vi`?

Process attributes

For every process that is created the UNIX operating system stores information including

- its real UID, GID and its effective UID and GID
  These are used to identify the owner of the process (real UID and GID) and determine what the process is allowed to do (effective UID and GID)
- the code and variables used by the process (its address map)
- the status of the process
- its priority
- its parent process

Parent processes

All processes are created by another process (its parent). The creation of a child process is usually a combination of two operations

- forking
  A new process is created that is almost identical to the parent process. It will be using the same code.
- exec
  This changes the code being used by the process to that of another program.

When you enter a command it is the shell that performs these tasks. It will fork off a new process (which is running the shell's program). The child process then performs an `exec` to change to the code for the command you wish executed.

Examples of this are shown in the pstree section earlier in this chapter.

Process UID and GID

In order for the operating system to know what a process is allowed to do it must store information about who owns the process (UID and GID). The UNIX operating system stores two types of UID and two types of GID.

Real UID and GID

A process' real UID and GID will be the same as the UID and GID of the user who ran the process. Therefore any process you execute will have your UID and GID.

The real UID and GID are used for accounting purposes.

Effective UID and GID

The effective UID and GID are used to determine what operations a process can perform. In most cases the effective UID and GID will be the same as the real UID and GID.

However using special file permissions it is possible to change the effective UID and GID. How and why you would want to do this is examined later in this chapter. The following exercise asks you to create an executable program we will use to display the real and effective UID and GID.

Exercises

5.4. Create a text file called `i_am.c` that contains the following C program. Compile the program by using the following command
`cc i_am.cc -o i_am`
This will produce an executable program called `i_am`.
Run the program. (***rather than type the code, you should be able to cut and paste it from the online versions of this chapter that are on the CD-ROM and Web site***)

```
#include <stdio.h>
#include <unistd.h>  void main()
{
 int real_uid, effective_uid;
 int real_gid, effective_gid;   /* get the user id and
group id*/
 real_uid = getuid();
 effective_uid = geteuid();
 real_gid = getgid();
 effective_gid = getegid();   /* display what I found */
 printf( "The real uid is %d\n", real_uid );
 printf("The effective uid is %d\n", effective_uid );
 printf("The real gid is %d\n", real_gid );
 printf("The effective gid is %d\n", effective_gid );
}
```

5.5. Make sure you are logged in as a normal user when you start the following exercise. In a previous exercise you were asked to discover which user owns the /usr/sbin/atd and sendmail processes. Try to cause these programs to stop using the kill command. If it doesn't work, why not? There are two reasons which may explain this problem. What are they?

5.6. Use the ps command to discover which user is the "owner" of the atd and sendmail processes

Files

All the information stored by UNIX onto disk is stored in files. Under UNIX even directories are just special types of files. A previous reading has already introduced you to the basic UNIX directory hierarchy. The purpose of this section is to fill in some of the detail including discussion of

- file types
  UNIX recognises a number of special file types which are used for specific purposes (e.g. a directory is a special file type).
- Normal files
  Normal files, those used to store data, can also have a number of types which describe the type of data stored in the file (e.g. a GIF file, a Word document)
- file attributes
  The file type is just one of the attributes UNIX stores about files. There are many others including owner and size.

File types

UNIX supports a small number of different file types. The following table summarises these different file types. What the different file types are and what their purpose is will be explained as we progress. File types are signified by a single character which is used in the output of the ls command (you use the ls command to view the various attributes of a file)..

File type	Meaning
`-`	a normal file
`d`	a directory
`l`	symbolic link
`b`	block device file
`c`	character device file
`p`	a fifo or named pipe

Table 5.1
UNIX file types

For current purposes you can think of these file types as falling into three categories

- normal ”files,
  Normal files are used to store data and under UNIX they are just a collection of bytes of information. The format of these bytes can be used to identify a normal file as a GIF file or a Word document.
- directories or directory files,
  Remember, for UNIX a directory is just another file which happens to contain the names of files and their I-node. An I-node is an operating system data structure which is used to store information about the file (explained later).
- special or device files.
  Explained in more detail later on in the text these special files provide access to devices which are connected to the computer. Why these exist and what they are used for will be explained.

Types of normal files

Those of you who use Windows will be familiar with normal files having different types (e.g. GIF images, Word documents etc). Under Windows the type of a normal file is specified by its extension. UNIX does not use this approach. In fact the operating system makes no distinction between different types of files. All files are simply a collection of bytes.

However, UNIX does provide commands which allow you to determine the type of normal files. If you''re unsure what type of normal file you have the UNIX `file` command might help.

```
[david@faile david]$ file article.doc reopen.call gtop.gif pair.pdf
/etc/passwd
article.doc:          Microsoft Office Document
reopen.call:          Microsoft Office Document
gtop.gif:             GIF image data, version 89a, 618 x 428,
pair.pdf:             PDF document, version 1.2
/etc/passwd:          ASCII text
```

In this example the `file` command has been used to discover what type of file for a number of files. Some important things to notice

- extension doesn't matter
  The file reopen.call is a Word document but its extension is not .doc.
- Additional features
  For some file types the file command provides additional features such as the height and width of the GIF image and the version of PDF used in the PDF file.

How does the file command work?

The `file` command attempts to perform three tests on a file to determine its type. The first test which works is used. The three tests are

file system tests
This works if the file to be tested is one of the special files listed in the previous section (e.g. a directory, device file etc). For example

```
[david@faile 2000]$ file /home /dev/hda
/home:    directory
/dev/hda: block special (3/0)
```

magic number tests
Many data file formats always contain a specific value at a specific location in the file. These value is referred to as a magic number. UNIX systems maintain a data file (/usr/share/magic on Linux) which contains a collection of magic numbers for various file types (take a look at the file on your Linux computer).

language tests
Finally if the file is a text file it attempts to determine what type of computer language the file contains.

Exercises

5.7. Examine the contents of the `/usr/share/magic` file. Experiment with the file command on a number of different files.

File attributes

The UNIX/Linux operating system uses a data structure called an inode to store all of the information it stores about a file (except for the filename). Every file on a UNIX system must have an associated inode on the disk. If you run out of inodes you can't create any more files on that disk.

You can find out which inode a file has by using the `ls -i` command.

```
dinbig:~$ ls -i README
  45210 README
```

In the above example the file README is using inode 45210.

Some fo the information UNIX stores about each file includes

- where the file's data is stored on the disk
  This is the collection of disk blocks which are used to store the data for the file.
- what the file's name is
  The name of a file is actually stored in the directory in which it occurs. Each entry in a directory contains a filename and an associated inode number.
- who owns the file
  The inode contains the UID and GID of the user who owns the file.
- who is allowed to do what with the file
  This is stored in the file permissions of a file. We examine file permissions in more detail below.
- how big the file is
- when was the file last modified
- how many links there are to the file
  It is possible for the same file to be known by more than one name. Remember the filename is stored in a directory. It is possible to have many different directories contain pointers to the one inode.

Throughout this text you will find the term file used to mean both files and directories.

Viewing file attributes

To examine the various attributes associated with a file you can use the `-l` switch of the `ls` command. This section provides some additional information about the file attributes.

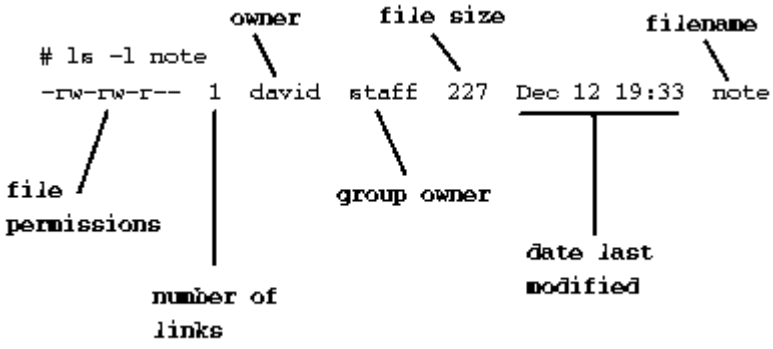


Figure 5.1
File Attributes

Filenames

Most UNIX file systems (including the Linux file system) will allow filenames to be 255 characters long and use almost any characters. However there are

some characters that can cause problems if used including `* $ ? ' " / \ -` and others (including the space character). Why is explained in the next chapter. This doesn't mean you can't create filenames that contain these characters, just that you can have some problems if you do.

Size

The size of a file is specified in bytes. So the above file is 227 bytes long. The standard Linux file system, called EXT2, will allow files to be up to 2Gb (giga bytes) in size. Which these days is not that large. Development is currently on-going for EXT3 which will increase this file size.

Owner and Group Owner

Even though the ls command displays the names of the user and group owner of a file that is not what is stored on the inode. The main reason being is that it would consume too much space to store the names. Instead the inode contains the UID and GID of the user and group owner. The ls command performs a translation from UID/GID to the name when it executes.

Date

The date specified here is the date the file was last modified.

Permissions

The permission attributes of a file specifies what operations can be done with a file and who can perform those operations. Permissions are explained in more detail in the following section.

Exercises

5.8. Examine the following command and it's output (executing these commands on your system should provide very similar results).

```
[david@faile 3]$ ls -ld / /dev
drwxr-xr-x 19 root root 1024 Dec 6 11:30 /
drwxr-xr-x 2 root root 22528 Dec 8 10:12 /dev
```

Answer the following questions

1. What type of files are / and /dev?
2. What else can you tell about these files?
3. How come /dev is bigger than /?

5.9. Execute the following commands

```
mkdir tmp
ls -ld tmp
touch tmp/tempfiledj
ls -ld tmp
```

These commands create a new directory called `tmp` and create an empty file `tempfiledj` inside that directory. The `touch` command is used to create an empty file if the file doesn't exist, or updates the date last modified if it does.

Why does the output of the `ls -ld tmp` command change?

File protection

Given that there can be many people sharing a UNIX computer it is important that the operating system provide some method of restricting access to files. I don't want you to be able to look at my personal files.

UNIX achieves this by

- restricting users to three valid operations,
  Under UNIX there are only three things you can do to a file (or directory): read, write or execute it.
- allow the file owner to specify who can do these operations on a file.
  The file owner can use the user and group concepts of UNIX to restrict which users (actually it restricts which processes that are owned by particular users) can perform these tasks.

File operations

UNIX provides three basic operations that can be performed on a file or a directory. The following table summarises those operations.

It is important to recognise that the operations are slightly different depending whether they are being applied to a file or a directory.

Operation	Effect on a file	Effect on a directory
read	read the contents of the file	find out what files are in the directory, e.g. `ls`
write	delete the file or add something to the file	be able to create or remove a file from the directory
execute	be able to run a file/program	be able to access a file within a directory

Table 5.2
UNIX file operations

Users, groups and others

Processes wishing to access a file on a UNIX computer are placed into one of three categories

- **user**
  The individual user who owns the file (by default the user that created the file but this can be changed). In figure 5.1 the owner is the user `david`.
- **group**
  The collection of people that belong to the group that owns the file (by default the group to which the file's creator belongs). In figure 5.1 the group is `staff`.
- **other**
  Anybody that doesn't fall into the first two categories.

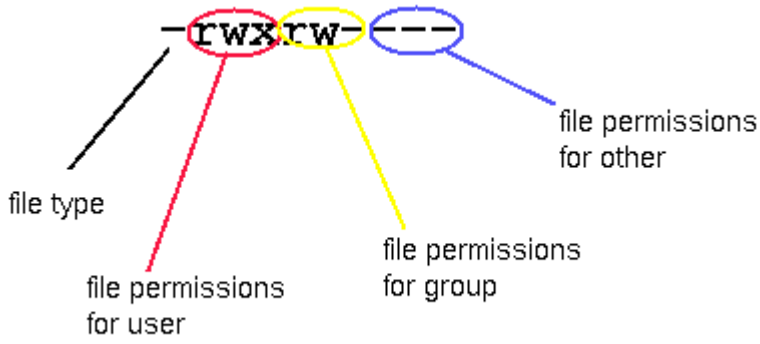


Figure 5.2

File Permissions

File permissons

Each user category (user, group and other) have their own set of file permissions. These control what file operation each particular user category can perform.

File permissions are the first field of file attributes to appear in the output of `ls -l`. File permissions actually consist of four fields

- file type,
- user permissions,
- group permissions,
- and other permissions.

Three sets of file permissions

As Figure 5.2 shows the file permissions for a file are divided into three different sets one for the user, one for a group which owns the file and one for everyone else.

A letter indicates that the particular category of user has permission to perform that operation on the file. A `-` indicates that they can't.

In the above diagram the owner can read, write and execute the file (`rwx`). The group can read and write the file (`rw-`), while other cannot do anything with the file (`---`).

Symbolic and numeric permissions

`rwxr-x-w-` is referred to as symbolic permissions. The permissions are represented using a variety of symbols.

There is another method for representing file permissions called numeric or absolute permissions where the file permissions are represented using numbers. The relationship between symbolic and numeric permissions is discussed in a couple of pages.

Symbols

The following table summarises the symbols that can be used in representing file permissions using the symbolic method.

Symbol	Purpose
`r`	read
`w`	write
`x`	execute
`s`	setuid or setgid (depending on location)
`t`	sticky bit

Table 5.3
Symbolic file permissions

Special permissions

Table 5.3 introduced three new types of permission setuid, setgid and the sticky bit.

Sticky bit on a file

In the past having the sticky bit set on a file meant that when the file was executed the code for the program would "stick" in RAM. Normally once a program has finished its code was taken out of RAM and that area used for something else.

The sticky bit was used on programs that were executed regularly. If the code for a program is already in RAM the program will start much quicker because the code doesn't have to be loaded from disk.

However today with the advent of shared libraries and cheap RAM most modern Unices ignore the sticky bit when it is set on a file.

Sticky bit on a directory

The `/tmp` directory on UNIX is used by a number of programs to store temporary files regardless of the user. For example when you use `elm` (a UNIX mail program) to send a mail message, while you are editing the message it will be stored as a file in the `/tmp` directory.

Modern UNIX operating systems (including Linux) use the sticky bit on a directory to make `/tmp` directories more secure. Try the command `ls -ld /tmp` what do you notice about the file permissions of `/tmp`.

If the sticky bit is set on a directory you can only delete or rename a file in that directory if you are

- the owner of the directory,
- the owner of the file, or
- the super user

Effective UID and GID

In this section we revisit the discussion of the relationship between the process attributes of real UID/GID and effective UID/GID.

When you use the `passwd` command to change your password the command will actually change the contents of either the `/etc/passwd` or `/etc/shadow` files. These are the files where your password is stored. By default most Linux systems use `/etc/passwd`

As has been mentioned previously the UNIX operating system uses the effective UID and GID of a process to decide whether or not that process can modify a file. Also the effective UID and GID are normally the UID and GID of the user who executes the process.

This means that if I use the `passwd` command to modify the contents of the `/etc/passwd` file (I write to the file) then I must have write permission on the `/etc/passwd` file. Let's find out.

What are the file permissions on the `/etc/passwd` file?

```
dinbig:~$ ls -l /etc/passwd
-rw-r--r--   1 root     root          697 Feb  1 21:21 /etc/passwd
```

On the basis of these permissions should I be able to write to the `/etc/passwd` file?

No. Only the user who owns the file, `root`, has write permission. Then how do does the `passwd` command change my password?

setuid and setgid

This is where the setuid and setgid file permissions enter the picture. Let's have a look at the permissions for the `passwd` command (first we find out where it is).

```
dinbig:~$ which passwd
/usr/bin/passwd
dinbig:~$ ls -l /usr/bin/passwd
-rws--x--x   1 root     bin          7192 Oct 16 06:10
/usr/bin/passwd
```

Notice the `s` symbol in the file permissions of the `passwd` command, this specifies that this command is setuid.

The setuid and setgid permissions are used to change the effective UID and GID of a process. When I execute the `passwd` command a new process is created. The real UID and GID of this process will match my UID and GID. However the effective UID and GID (the values used to check file permissions) will be set to that of the command.

In the case of the `passwd` command the effective UID will be that of `root` because the setuid permission is set, while the effective GID will be my group's because the setgid bit is not set.

Exercises

5.10. Log in as the root user, go to the directory that contains the file `i_am` you created in exercise 5.3. Execute the following commands

```
cp i_am i_am_root
cp i_am i_am_root_group
chown root.root i_am_root*
chmod a+rx i_am*
chmod u+s i_am_root
chmod +s i_am_root_group
ls -l i_am*
```

These commands make copies of the `i_am` program called `i_am_root` with setuid set, and `i_am_root_group` with setuid and setgid set. Log back in as your normal user and execute all three of the `i_am` programs. What do you notice? What is the UID and gid of `root`?

Numeric permissions

Up until now we have been using symbols like `r w x s t` to represent file permissions. However the operating system itself doesn't use symbols, instead it uses numbers. When you use symbolic permissions, the commands translate between the symbolic permission and the numeric permission.

With numeric or absolute permissions the file permissions are represented using octal (base 8) numbers rather than symbols. The following table summarises the relationship between the symbols used in symbolic permissions and the numbers used in numeric permissions.

To obtain the numeric permissions for a file you add the numbers for all the permissions that are allowed together.

Symbol	Number
s	4000 setuid 2000 setgid
t	1000
r	400 user 40 group 4 other
w	200 user 20 group 2 other
x	100 user 10 group 1 other

Table 5.4
Numeric file permissions

Symbolic to numeric

Here's an example of converting from symbolic to numeric using a different method. This method relies on using binary numbers to calculate the numeric permissions.

The process goes something like this

- write down the symbolic permissions,
- under each permission that is on, write a one
- under each permission that is off, write a zero
- for each category of user, user, group and other convert the three binary digits into decimal, e.g. rwx -> 111 -> 7
- combine the three numbers (one each for user, group and other) into a single octal number

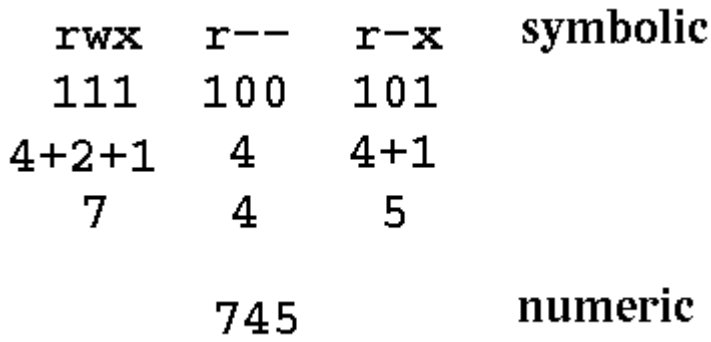


Figure 5.3
Symbolic to Numeric permissions

Exercises

5.11. Convert the following symbolic permissions to numeric

```
rwxrwxrwx
---------
---r--r--
r-sr-x---
rwsrwsrwt
```

5.12. Convert the following numeric permissions to symbolic

```
710
4755
5755
6750
7000
```

Changing file permissions

The UNIX operating system provides a number of commands for users to change the permissions associated with a file. The following table provides a summary.

Command	Purpose
chmod	change the file permissions for a file
umask	set the default file permissions for any files to be created. Usually run as the user logs in.
chgrp	change the group owner of a file
chown	change the user owner of a file.

Table 5.5
Commands to change file ownership and permissions

chmod

The `chmod` command is used to the change a file's permissions. Only the user who owns the file can change the permissions of a file (the root user can also do it).

Format

`chmod [-R] operation files`

The optional (the `[ ]` are used to indicate optional) switch `-R` causes `chmod` to recursively descend any directories changing file permissions as it goes.

`files` is the list of files and directories to change the permissions of.

`operation` indicates how to change the permissions of the files. *`operation`* can be specified using either symbolic or absolute permissions.

Numeric permissions

When using numeric permissions *`operation`* is the numeric permissions to change the files permissions to. For example

```
chmod 770 my.file
```

will change the file permissions of the file `my.file` to the numeric permissions `770`.

Symbolic permissions

When using symbolic permissions *`operation`* has three parts *`who op symbolic_permission`* where

- *`who`* specifies the category of user to change the permissions for
  It can be any combination of `u` for user, `g` for group, `o` for others and `a` for all categories.

- *op* specifies how to change the permissions
  + add permission, - remove permission, = set permission
- *permission* specifies the symbolic permissions
  r for read, w for write, x execute, s set uid/gid, t set sticky bit.

Examples

- `chmod u+rwx temp.dat`
  add rwx permission for the owner of the file, these permissions are added to the existing permissions
- `chmod go-rwx temp.dat`
  remove all permissions for the group and other categories
- `chmod -R a-rwx /etc`
  turn off all permissions, for all users, for all files in the `/etc` directory.
- `chmod -R a= /`
  turn off all permissions for everyone for all files
- `chmod 770 temp.dat`
  allow the user and group read, write and execute and others no access

chown

The UNIX operating system provides the `chown` command so that the owner of a file can be changed. However in most Unices only the root user can use the command.

Two reasons why this is so are

- in a file system with quotas (quotas place an upper limit of how many files and how much disk space a user can use) a person could avoid the quota system by giving away the ownership to another person
- if anyone can give ownership of a file to root they could create a program that is setuid to the owner of the file and then change the owner of the file to root

chgrp

UNIX also supplies the command `chgrp` to change the group owner of a file. Any user can use the `chgrp` command to change any file they are the owner of. However you can only change the group owner of a file to a group to which you belong.

For example

```
dinbig$ whoami
david
dinbig$ groups
users
dinbig$ ls -l tmp
-rwxr-xr-x 2 david users 1024 Feb 1 21:49 tmp
dinbig$ ls -l /etc/passwd
```

```
dinbig$ chgrp users /etc/passwd
chgrp: /etc/passwd: Operation not permitted
-rw-r--r-- 1 root root 697 Feb 1 21:21 /etc/passwd
dinbig$ chgrp man tmp
chgrp: you are not a member of group 'man': Operation not permitted
```

In this example I've tried to change the group owner of `/etc/passwd`. This failed because I am not the owner of that file.

I've also tried to change the group owner of the file `tmp`, of which I am the owner, to the group `man`. However I am not a member of the group `man` so it has also failed.

chown and chgrp

The commands `chown` and `chgrp` are used to change the owner and group owner of a file.

Format

```
chown [-R] owner files
chgrp [-R] group files
```

The optional switch `-R` works in the same was as the `-R` switch for chmod. It modifies the command so that it descends any directories and performs the command on those sub-directories and files in those sub-directories.

`owner` is either a numeric user identifier or a username.

`group` is either a numeric group identifier or a group name.

`files` is a list of files of which you wish to change the ownership.

Some systems (Linux included) allow *`owner`* in the `chown` command to take the format *`owner.group`*. This allows you to change the owner and the group owner of a file with one command.

Examples

- `chown david /home/david`
  Change the owner of the directory `/home/david` to `david`. This demonstrates one of the primary uses of the `chown` command. When a new account is created the root user creates a number of directories and files. Since root created them they are owned by root. In real life these files and directories should be owned by the new username.
- `chown -R root /`
  Change the owner of all files to `root`.
- `chown david.users /home/david`
  Change the ownership of the file `/home/david` so that it is owned by the user `david` and the group `users`.
- `chgrp users /home/david`
  Change the group owner of the directory `/home/david` to the group `users`.

Default permissions

When you create a new file it automatically receives a set of file permissions.

```
dinbig:~$ touch testing
dinbig:~$ ls -l testing
-rw-r--r--   1 david    users           0 Feb 10 17:36 testing
```

In this example the touch command has been used to create an empty file called testing. The touch command usually updates the last modified date of a file to the current time. However, if the file doesn't exist it creates an empty file of that same name.

In this example the file `testing` has been given the default permissions `rw-r--r--`. Any file I create will receive the same default permissions.

`umask`

The built-in shell command `umask` is used specify and view what the default file permissions are. Executing the `umask` command without any arguments will cause it to display what the current default permissions are.

```
dinbig:~$ umask
022
```

By default the `umask` command uses the numeric format for permissions. It returns a number which specifies which permissions are turned **off** when a file is created.

In the above example

- the user has the value `0`
  This means that by default no permissions are turned off for the user.
- the group and other have the value `2`
  This means that by default the write permission is turned off.

You will notice that the even though the execute permission is not turned off my default file doesn't have the execute permission turned on. I am not aware of the exact reason for this.

`umask` versions

Since `umask` is a built-in shell command the operation of the `umask` command will depend on the shell you are using. This also means that you'll have to look at the man page for your shell to find information about the `umask` command.

`umask` for `bash`

The standard shell for Linux is `bash`. The version of `umask` for this shell supports symbolic permissions as well as numeric permissions. This allows you to perform the following.

```
dinbig:~$ umask -S
u=rwx,g=r,o=r
dinbig:~$ umask u=rw,g=rw,o=
```

```
dinbig:~$ umask -S
u=rw,g=rw,o=
```

Exercises

5.13.	Use the `umask` command so that the default permissions for new files are set to
	`rw-------`
	`772`

5.14.	How do you test that this is working correctly?

File permissions and directories

As shown in table 5.2 file permissions have a slightly different effect on directories than they do on files.

The following example is designed to reinforce your understanding of the effect of file permissions on directories.

For example

Assume that

- I have an account on the same UNIX machine as you
- we belong to different groups
- I want to allow you to access the text for assignment one
- I want you to copy your finished assignments into my directory
- But I don't want you to see anything else in my directories

The following diagram represents part of my directory hierarchy including the file permissions for each directory.

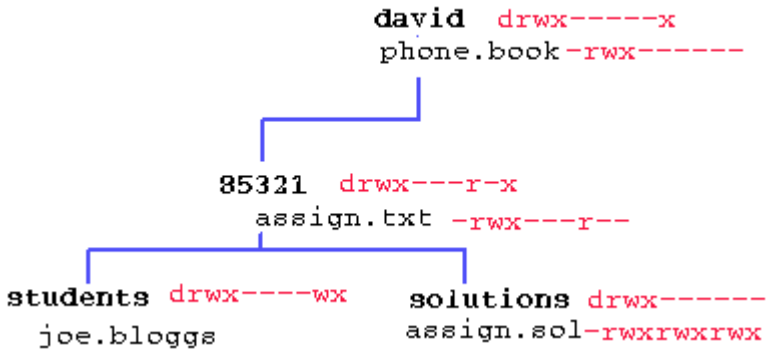


Figure 5.4
Permissions and Directories

What happens if?

What happens if you try the following commands

- `ls -l david`
  To perform an `ls` you must have read permission on the directory. In this case you don't. Only myself, as the owner of the file has read permission, so only I can obtain a listing of the files in my directory.

- `cat david/phone.book`
  You''re trying to have a look at my phone book but you can't. You have permission to do things to files in my directory because you have execute permission on the directory `david`. However the permissions on the `phone.book` file mean that only I can read it. The same things occurs if you try the command `cp david/phone.book ~/phone.book`. To the file system you are trying to do the same thing, read the file `phone.book`.

- `ls david/85321` The permissions are set up so you can get a listing of the files in the `david/85321` directory. Notice you have read permission on the `85321` directory.

- `cat david/85321/assign.txt`
  Here you're trying to have a look at the assignment text. This will work. You have read permission on the file so you can read it. You have execute permission on the directories `85321` and `david` which means you can gain access to files and directories within those directories (if the permissions on the files let you).

- `cat david/85321/solutions/assign1.sol`
  Trying to steal a look at the solutions? Well you have the permissions on the file to do this. But you don't have the permissions on the directory `solutions` so this will fail.
  What would happen if I executed this command

- `chmod o+r david/85321/solutions`
  This would add read permission for you to the directory `solutions`. Can you read the `assign1.sol` file now? No you can't. To read the file or do anything with a file you must have execute permission on the directory it is in.

- `cp my.assign david/85321/assign.txt`
  What's this? Trying to replace my assignment with one of your own? Will this work? No because you don't have write permission for the file `assign.txt`.

Links

As mentioned earlier in this chapter it is possible for a single UNIX file to be known by more than one name. This is achieved using links. This section provides a brief overview of the two types of links supported by Linux, hard and soft.

Links are useful for a number of reasons

- simplify file paths
  Rather than ask users to remember that a certain data file is stored in the file

`/usr/local/www/data/1999/T3/85349/assign1/solutions.txt` you can create a link `/85349/a1/solutions.txt` which be used by people.

- Support different locations
  Some versions of UNIX and UNIX commands expect files or directories to be in different locations. For example, some expect the executable program sendmail to be in `/usr/lib/sendmail` while others expect it to be in `/usr/sbin/sendmail`. Under Linux `/usr/lib/sendmail` is a hard link to `/usr/sbin/sendmail`.

The idea behind links is that you can use a different name to refer to the same file without having to duplicate the data blocks for the file.

Creating Links

Links, both hard and soft links, are created using the ln command. By default ln creates hard links. The ln command has two main formats

- ln *target* [*link-name*]
  Where *target* specifies the name of the file being linked to and *link-name* is the optional name of the link to create. Some examples

```
[david@faile tmp]$ ln /etc/passwd
[david@faile tmp]$ ln /etc/passwd fred
[david@faile tmp]$ ls -il * /etc/passwd
 308647 -rw-r--r--   3 root     root          681 Jan  3 08:37
/etc/passwd
 308647 -rw-r--r--   3 root     root          681 Jan  3 08:37 fred
 308647 -rw-r--r--   3 root     root          681 Jan  3 08:37 passwd
```

With the first ln command the *link-name* is not specified. In this instance the ln command creates a hard link with the same name as the target (passwd). In the second example the *link-name* is specified. The output of the ls command shows an important point about hard links (remember by default ln creates hard links). Since I am using the -i switch of the ls command we are shown the inodes for the three files. Each of these filenames (/etc/passwd, fred and passwd) all point to the same file and so they have the same inode.

- ln *target-list directory*
  In this format the ln command allows you to specify multiple targets followed by a directory name. The end result is that the directory will end up containing hard links to each of the targets. The names of the hard links will match that of the targets. For example,

```
[david@faile tmp]$ ln /etc/passwd /usr/share/magic /etc/group .
[david@faile tmp]$ ls -il
total 184
 308651 -rw-r--r--   2 root     root          459 Oct  7 14:24 group
 227274 -rw-r--r--   2 root     root       173852 Aug 24 08:06 magic
 308647 -rw-r--r--   2 root     root          681 Jan  3 08:37 passwd
[david@faile tmp]$ ls -il /etc/passwd /usr/share/magic /etc/group
 308651 -rw-r--r--   2 root     root          459 Oct  7 14:24 /etc/group
 308647 -rw-r--r--   2 root     root          681 Jan  3 08:37 /etc/passwd
 227274 -rw-r--r--   2 root     root       173852 Aug 24 08:06
/usr/share/magic
```

In this example there are three targets, /etc/passwd /usr/share/magic and /etc/group, and the directory specified is . (the full stop character which indicates the current directory). Again you can see the connection between the targets and the hard links via the similarity in the inodes.

The ln command supports a number of command-line switches which can be used to modify its operation in a number of different ways. The options include

- -b, the backup switch
  If there is already a file with the same name as the link this switch will make a backup of the existing file.
- -s, the soft link switch
  This forces ln to create a soft link rather than a hard link.

The manual page for the ln command explains further. Some examples

```
[david@faile tmp]$ ln -b /etc/group passwd
[david@faile tmp]$ ln -sb /etc/group magic
[david@faile tmp]$ ls -il
total 188
 308651 -rw-r--r--   3 root     root          459 Oct  7 14:24 group
1105693 lrwxrwxrwx   1 david    david          10 Jan  9 14:34 magic ->
/etc/group
 227274 -rw-r--r--   2 root     root       173852 Aug 24 08:06 magic~
 308651 -rw-r--r--   3 root     root          459 Oct  7 14:24 passwd
 308647 -rw-r--r--   2 root     root          681 Jan  3 08:37 passwd~
```

The first thing to notice in the output of the ls command are the two new files magic~ and passwd~. These are the backup files which were created by using the -b switch of the ln command. The ln command creates backups by adding a ~ to the filename.

The other difference to notice is magic -> /etc/group output for the magic file. This is how the ls command indicates a soft link. Also notice how the file type for the symbolic link is now l.

Hard and soft links, the differences

The differences between hard and soft links can be placed into two categories

- implementation, and
- operational.

The implementation details will be discussed in more detail in the File System chapter later in the text and involves how the operating system actually implements hard and soft links. One of the major implementation differences is that hard links and the file they are pointing to refer to the same inode where as soft links have their own inodes.

The difference in inodes is the reason behind the operational differences between the two types of links. Standard file operations which use or modify data which is stored on the inode will behave differently with soft links than they do with hard links. Remember soft links use different inodes. So if you change any information on the inode of a soft link that same change will not be made on the inode of the file the soft link is pointing to. However, making any

change to the inode information of a hard link also changes the inode information for the file the link is pointing to.

Earlier in this chapter we introduced the information which is stored on the inode including file permissions, modified date, file size etc. Let's create some links and compare the results.

```
[david@faile tmp]$ ln -s ../rec/file.jpg file
[david@faile tmp]$ ln ../rec/anna.jpg anna
[david@faile tmp]$ ls -il file anna ../rec/anna.jpg ../rec/file.jpg
 537951 -rw-rw-r--   2 david    david        5644 Dec 20 19:13
../rec/anna.jpg
 537955 -rwxrwxr-x   1 david    david       17833 Dec 20 19:20
../rec/file.jpg
 537951 -rw-rw-r--   2 david    david        5644 Dec 20 19:13 anna
1105712 lrwxrwxrwx   1 david    david          15 Jan  9 15:19 file ->
../rec/file.jpg
```

Notice how the information displayed about the hard link (anna) and the file it points to (../rec/anna.jpg) are exactly the same. This idicates a hard link. Notice how the information about the soft link (file) and the file it points to (../rec/file.jpg) are different?

```
[david@faile tmp]$ chmod a+x anna
[david@faile tmp]$ chmod a-x file
[david@faile tmp]$ ls -il file anna ../rec/anna.jpg ../rec/file.jpg
 537951 -rwxrwxr-x   2 david    david        5644 Dec 20 19:13
../rec/anna.jpg
 537955 -rw-rw-r--   1 david    david       17833 Dec 20 19:20
../rec/file.jpg
 537951 -rwxrwxr-x   2 david    david        5644 Dec 20 19:13 anna
1105712 lrwxrwxrwx   1 david    david          15 Jan  9 15:19 file ->
../rec/file.jpg
```

The chmod command changes the permissions associated with a file. From the above you can see that if you change the file permissions for a hard link that the permissions for both the hard link and the file it is pointing to change. However for the software link only the permissions for the file being pointed to change.

Another important difference between hard and soft links is what happens when you start deleting files.

```
[david@faile tmp]$ rm ../rec/file.jpg ../rec/anna.jpg
[david@faile tmp]$ ls -il file anna ../rec/anna.jpg ../rec/file.jpg
ls: ../rec/anna.jpg: No such file or directory
ls: ../rec/file.jpg: No such file or directory
 537951 -rwxrwxr-x   1 david    david        5644 Dec 20 19:13 anna
1105712 lrwxrwxrwx   1 david    david          15 Jan  9 15:19 file ->
../rec/file.jpg
[david@faile tmp]$ file anna file
anna: JPEG image data, JFIF standard
file: broken symbolic link to ../rec/file.jpg
```

When you remove a hard link or the file it points to it simply reduces the link count. The link count is the first number after the file permissions. Notice how the file anna has a link count of 1 in the above example where it had a 2 in earlier examples. The file anna still has a link to the data files contained in the image anna.jpg.

On the other hand when you remove the file a soft link points to you are in trouble. The symbolic link does not know that the target has disappeared. The output of the file command shows this.

Searching the file hierarchy

A common task for a Systems Administrator is searching the UNIX file hierarchy for files which match certain criteria. Some common examples of what and why a Systems Administrator may wish to do this include

- searching for very large files
- finding where on the disk a particular file is
- deleting all the files owned by a particular user
- displaying the names of all files modified in the last two days.

Given the size of the UNIX file hierarchy and the number of files it contains this isn't a task that can be done by hand. This is where the `find` command becomes useful.

The `find` command

The `find` command is used to search through the directories of a file system looking for files that match a specific criteria. Once a file matching the criteria is found the `find` command can be told to perform a number of different tasks including running any UNIX command on the file.

`find` command format

The format for the `find` command is

```
find [path-list] [expression]
```

`path-list` is a list of directories in which the `find` command will search for files. The command will recursively descend through all sub-directories under these directories. The expression component is explained in the next section.

Some examples

```
[root@faile tmp]# find /etc -type s
```

The path-list contains just /etc while the expression is -type s. In this case there are no files underneath the /etc directory which match the expression.

```
[root@faile tmp]# find /etc /usr/share -name magic -o -name passwd
/etc/passwd
/etc/pam.d/passwd
/etc/uucp/passwd
/usr/share/magic
```

In this example the path-list contains the directories /etc and /usr/share and the expression is -name magic -o -name passwd. This command finds four files under either the /etc or /usr/share directories which have the filename passwd or magic.

Both the path and the expression are optional. If you run the `find` command without any parameters it uses a default path, the current directory, and a default expression, print the name of the file. The following is an example of what happens

```
dinbig:~$ find
.
./iAm
./iAm.c
./parameters
./numbers
./pass
./func
./func2
./func3
./pattern
./Adirectory
./Adirectory/oneFile
```

The default path is the current directory. In this example the `find` command has recursively searched through all the directories within the current directory.

The default expression is `-print`. This is a `find` command that tells the `find` command to display the name of all the files it found.

Since there was no test specified the `find` command matched all files.

`find` expressions

A `find` expression can contain the following components

- options,
  These modify the way in which the `find` command operates.
- tests,
  These decide whether or not the current file is the one you are looking for.
- actions,
  Specify what to do once a file has been selected by the tests.
- and operators.
  Used to group expressions together.

`find` options

Options are normally placed at the start of an expression. Table 5.6 summarises some of the `find` commands options.

Option	Effect
`-daystart`	for tests using time measure time from the beginning of today
`-depth`	process the contents of a directory before the directory
`-maxdepth` *`number`*	*`number`* is a positive integer that specifies the maximum number of directories to descend
`-mindepth` *`number`*	*`number`* is a positive integer that specifies at which level to start applying tests
`-mount`	don't cross over to other partitions
`-xdev`	don't cross over to other partitions

Table 5.6
`find` options

For example

The following are two examples of using `find`'s options. Since I don't specify a path in which to start searching the default value, the current directory, is used.

```
dinbig:~$ find -mindepth 2
./Adirectory/oneFile
```

In this example the `mindepth` option tells find to only find files or directories which are at least two directories below the starting point.

```
dinbig:~$ find -maxdepth 1
.
./iAm
./iAm.c
./parameters
./numbers
./pass
./func
./func2
./func3
./pattern
./Adirectory
```

This option restricts `find` to those files which are in the current directory.

`find` tests

Tests are used to find particular files based on

- when the file was last accessed
- when the file's status was last changed
- when the file was last modified
- the size of the file
- the file's type
- the owner or group owner of the file
- the file's name
- the file's inode number
- the number and type of links the file has to it
- the file's permissions

Table 5.7 summarises `find`'s tests. A number of the tests take numeric values. For example, the number of days since a file was modified. For these situations the numeric value can be specified using one of the following formats (in the following *n* is a number)

- *+n*
  greater than *n*
- *-n*
  less than *n*

- *n*
  equal to *n*

For example

Some examples of using tests are shown below. Note that in all these examples no command is used. Therefore the `find` command uses the default command which is to print the names of the files.

- `find . -user david`
  Find all the files under the current directory owned by the user `david`
- `find / -name \*.html`
  Find all the files one the entire file system that end in `.html`. **Notice** that the `*` must be quoted so that the shell doesn't interpret it (explained in more detail below). Instead we want the shell to pass the `*.html` to the find command and have it match filenames.
- `find /home -size +2500k -mtime -7`
  Find all the files under the `/home` directory that are greater than 2500 kilobytes in size and have been in modified in the last seven days.

The last example shows it is possible to combine multiple tests. It is also an example of using numeric values. The `+2500` will match any value greater than 2500. The `-7` will match any value less than 7.

Shell special characters

The shell is the program which implements the UNIX command line interface at which you use these commands. Before executing commands the shell looks for special characters. If it finds any it performs some special operations. In some cases, like the previous command, you don't want the shell to do this. So you quote the special characters. This process is explained in more detail in the following chapter.

Test	Effect
`-amin` *n*	file last access *n* minutes ago
`-anewer` *file*	the current file was access more recently than *file*
`-atime` *n*	file last accessed *n* days ago
`-cmin` *n*	file's status was changed *n* minutes ago
`-cnewer` *file*	the current file's status was changed more recently than *file*'s
`-ctime` *n*	file's status was last changed *n* days ago
`-mmin` *n*	file's data was last modified *n* minutes ago
`-mtime` *n*	the current file's data was modified *n* days ago
`-name` *pattern*	the name of the file matches *pattern* `-iname` is a case insensitive version of `–name` `-regex` allows the use of REs to match filename
`-nouser-nogroup`	the file's UID or GID does not match a valid user or group
`-perm` *mode*	the file's permissions match *mode* (either symbolic or numeric)
`-size` *n*`[bck]`	the file uses *n* units of space, `b` is blocks, `c` is bytes, `k` is kilobytes
`-type` *c*	the file is of type *c* where *c* can be `b`lock device file, `c`haracter device file, `d`irectory, named `p`ipe, regular `f`ile, symbolic `l`ink, `s`ocket
`-uid` *n* `-gid` *n*	the file's UID or GID matches *n*
`-user` *uname*	the file is owned by the user with name *uname*

Table 5.7
`find` tests

`find` actions

Once you've found the files you were looking for you want to do something with them. The `find` command provides a number of actions most of which allow you to either

- execute a command on the file, or
- display the name and other information about the file in a variety of formats

For the various `find` actions that display information about the file you are urged to examine the manual page for `find`

Executing a command

`find` has two actions that will execute a command on the files found. They are `-exec` and `-ok`.

The format to use them is as follows

```
-exec command ;
-ok command ;
```

command is any UNIX command.

The main difference between `exec` and `ok` is that `ok` will ask the user before executing the command. `exec` just does it.

For example

Some examples of using the `exec` and `ok` actions include

- `find . -exec grep hello \{\} \;`
  Search all the files under the local directory for the word `hello`.
- `find / -name \*.bak -ok rm \{\} \;`
  Find all files ending with `.bak` and ask the user if they wish to delete those files.

`{}` and `;`

The `exec` and `ok` actions of the `find` command make special use of `{}` and `;` characters. All the characters here ({ and } and ;) all have special meaning to the shell and as a result they must be quoted when used with the `find` command.

The find command uses `{}` to refer to the filename that `find` has just tested. So in the last example `rm \{\}` will delete each file that the `find` tests match.

The `;` is used to indicate the end of the command to be executed.

Exercises

5.15. Use `find` to print the names of every file on your file system that has nothing in it find where the file `XF86Config` is

5.16. Write find commands to
- find all the files called core and display their full details
- find all the files called core which haven't been accessed in the last week and delete them
- find all the files which are greater than 1Mb in size, were created in the last month and are not owned by the root user

5.17. Write find commands to
- find all the files called core and display their full details
- find all the files called core which haven't been accessed in the last week and delete them
- find all the files which are greater than 1Mb in size, were created in the last month and are not owned by the root user

Performing commands on many files

Every UNIX command you execute requires a new process to be created. Creating a new process is a fairly heavyweight procedure for the operating system and can take quite some time. When you are performing a task it can save time if you minimise the number of new processes which are created.

It is common for a Systems Administrator to want to perform some task which requires a large number of processes. Some uses of the `find` command offer a good example.

For example

Take the requirement to find all the HTML files on a Web site which contain the word `expired`. There are at least three different ways we can do this

- using the `find` command and the `-exec` switch,
- using the `find` command and back quotes ‘‘,
- using the `find` command and the `xargs` command.

In the following we’ll look at each of these.

> **More than one way to do something**
>
> One of the characteristics of the UNIX operating system is that there is always more than one way to perform some task.

`find` and `-exec`

We’ll assume the files we are talking about in each of these examples are contained in the directory `/usr/local/www`

```
find /usr/local/www -name \*.html -exec grep -l expired \{\} \;
```

The `-l` switch of `grep` causes it to display the filename of any file in which it finds a match. So this command will list the names of all the files containing `expired`.

While this works there is a slight problem, it is inefficient. These commands work as follows

- `find` searches through the directory structure,
- every time it finds a file that matches the test (in this example that it has the extension `html`) it will run the appropriate command
- the operating system creates a new process for the command,
- once the command has executed for that file it dies and the operating system must clean up,
- now we restart at the top with `find` looking for the appropriate file

On any decent Web site it is possible that there will be tens and even hundreds of thousands of HTML files. For example, the 1999 website for 85321 contained 11,051 HTML files. This implies that this command will result in

hundreds of thousands of processes being created. This can take quite some time.

`find` and back quotes

A solution to this is to find all the matching files first, and then pass them to a single `grep` command.

```
grep -l expired ‘find /usr/local/www -name \*.html‘
```

In this example there are only two processes created. One for the `find` command and one for the `grep`.

> **Back quotes**
>
> Back quotes ‘‘ are an example of the shell special characters mentioned previously. When the shell sees ‘‘ characters it knows it must execute the command enclosed by the ‘‘ and then replace the command with the output of the command.
>
> In the above example the shell will execute the `find` command which is enclosed by the ‘‘ characters. It will then replace the *‘find /usr/local/www -name *.html‘* with the output of the command. Now the shell executes the grep command.
>
> Back quotes are explained in more detail in the next chapter.

To show the difference that this makes you can use the `time` command. `time` is used to record how long it takes for a command to finish (and a few other stats). The following is an example from which you can see the significant difference in time and resources used by reducing the number of processes.

```
beldin:~$ time grep -l expired ‘find 85321/* -name index.html‘
0.04user 0.22system 0:02.86elapsed 9%CPU (0avgtext+0avgdata
0maxresident)k 0inputs+0outputs (0major+0minor)pagefaults 0swaps
beldin:~$ time find 85321/* -name index.html -exec grep -l expired
\{\} \;
1.33user 1.90system 0:03.55elapsed 90%CPU (0avgtext+0avgdata
0maxresident)k 0inputs+0outputs (0major+0minor)pagefaults 0swaps
```

The `time` command can also report a great deal more information about a process and its interaction with the operating system. Especially if you use the verbose option (`time –v` *`some_command`*)

`find` and `xargs`

While in many cases the combination of `find` and back quotes will work perfectly, this method has one serious drawback as demonstrated in the following example.

```
beldin:~$ grep -l expired ‘find 85321/* -name \*‘
bash: /usr/bin/grep: Arg list too long
```

The problem here is that a command line can only be so long. In the above example the `find` command found so many files that the names of these files exceeded the limit.

This is where the `xargs` command enters the picture.

Rather than pass the list of filenames as a parameter to the command, `xargs` allows the list of filenames to be passed as standard input (standard input is explained in more detail in a following chapter). This means we side-step the problem of exceeding the allowed maximum length of a command line..

Have a look at the man page for `xargs` for more information. Here is the example rewritten to use `xargs`

```
find /usr/local/www -name \* | xargs grep -l expired
```

There are now three processes created, `find`, `xargs` and `grep`. However it does avoid the problem of the argument list being too long.

Conclusion

UNIX is a multi-user operating system and as such must provide mechanisms to uniquely identify users and protect the resources of one user from other users. Under UNIX users are uniquely identified by a username and a user identifier (UID). The relationship between username and UID is specified in the `/etc/passwd` file.

UNIX also provides the ability to collect users into groups. A user belongs to at least one group specified in the `/etc/passwd` file but can also belong to other groups specified in the `/etc/group` file. Each group is identified by both a group name and a group identifier (GID). The relationship between group name and GID is specified in the `/etc/group` file.

All work performed on a UNIX computer is performed by processes. Each process has a real UID/GID pair and an effective UID/GID pair. The real UID/GID match the UID/GID of the user who started the process and are used for accounting purposes. The effective UID/GID are used for deciding the permissions of the process. While the effective UID/GID are normally the same as the real UID/GID it is possible using the setuid/setgid file permissions to change the effective UID/GID so that it matches the UID and GID of the file containing the process' code.

The UNIX file system uses a data structure called an inode to store information about a file including file type, file permissions, UID, GID, number of links, file size, date last modified and where the files data is stored on disk. A file's name is stored in the directory which contains it.

A file's permissions can be represented using either symbolic or numeric modes. Valid operations on a file include read, write and execute. Users wishing to perform an operation on a file belong to one of three categories the user who owns the file, the group that owns the file and anyone (other) not in the first two categories.

A file's permissions can only be changed by the user who owns the file and are changed using the `chmod` command. The owner of a file can only be changed by the root user using the `chown` command. The group owner of a file can be

changed by root user or by the owner of the file using the `chgrp` command. The file's owner can only change the group to another group she belongs to.

Links both hard and soft are mechanisms by which more than one filename can be used to refer to the same file.

Review Questions

5.1

For each of the following commands indicate whether they are built-in shell commands, "normal" UNIX commands or not valid commands. If they are "normal" UNIX commands indicate where the command's executable program is located.

- alias
- history
- rename
- last

5.2

How would you find out what your UID, GID and the groups you currently belong to?

5.3

Assume that you are logged in with the username `david` and that your current directory contains the following files

```
bash# ls –il
total 2
103807 -rw-r--r-- 2 david users    0 Aug 25 13:24 agenda.doc
103808 -rwsr--r-- 1 root  users    0 Aug 25 14:11 meeting
103806 -rw-r--r-- 1 david users 2032 Aug 22 11:42 minutes.txt
103807 -rw-r--r-- 2 david users    0 Aug 25 13:24 old_agenda
```

For each of the following commands indicate

- whether or not it will work,
- if it works specify how the above directory listing will change,
- if it doesn't work why?

```
chmod 777 minutes.txt
chmod u+w agenda.doc
chmod o-x meeting
chmod u+s minutes.txt
ln -s meeting new_meeting
chown root old_agenda
```

5.4

Assume that the following files exist in the current directory.

```
bash$ ls -li
total 1
32845 -rw-r--r--  2 jonesd   users  0 Apr   6 15:38 cq_uni_doc
32845 -rw-r--r--  2 jonesd   users  0 Apr   6 15:38 cqu_union
32847 lrwxr-xr-x  1 jonesd   users  10 Apr  6 15:38 osborne ->
cq_uni_doc
```

For each of the following commands explain how the output of the command `ls -li` will change AFTER the command has been executed. Assume that that each command starts with the above information

For example, after the command `mv cq_uni_doc CQ.DOC` the only change would be that entry for the file `cq_uni_doc` would change to

```
32845 -rw-r--r--  2 jonesd   users  0 Apr   6 15:38 CQ.DOC
```

```
chmod a-x osborne
chmod 770 cqu_union
rm cqu_union
rm cqu_uni_doc
```

The files `cq_uni_doc` and `cqu_union` both point to the same file using a hard link. Above I have stated that if you execute the command `mv cq_uni_doc CQ.DOC` the only thing that changes is the name of the file `cq_uni_doc`. Why doesn't the name of the file `cqu_union` change also?

Chapter 6

The Shell

Introduction

You will hear many people complain that the UNIX operating system is hard to use. They are wrong. What they actually mean to say is that the UNIX command line interface is difficult to use. This is the interface that many people think is UNIX. In fact, this command line interface, provided by a program called a shell, is not the UNIX operating system and it is only one of the many different interfaces that you can use to perform tasks under UNIX. By this stage many of you will have used some of the graphical user interfaces provided by the X-Windows system.

The shell interface is a powerful tool for a Systems Administrator and one that is often used. This chapter introduces you to the shell, it's facilities and advantages. It is important to realise that the shell is just another UNIX command and that there are many different sorts of shell. The responsibilities of the shell include

- providing the command line interface
- performing I/O redirection
- performing filename substitution
- performing variable substitution
- and providing an interpreted programming language

The aim of this chapter is to introduce you to the shell and the first four of the responsibilities listed above. The interpreted programming language provided by a shell is the topic of chapter 8.

Executing Commands

As mentioned previously the commands you use such as `ls` and `cd` are stored on a UNIX computer as executable files. How are these files executed? This is one of the major responsibilities of a shell. The command line interface at which you type commands is provided by the particular shell program you are using (under Linux you will usually be using a shell called `bash`). When you type a command at this interface and hit enter the shell performs the following steps

- wait for the user to enter a command
- perform a number of tasks if the command contains any special characters
- find the executable file for the command, if the file can't be found generate an error message
- fork off a child process that will execute the command,

- wait until the command is finished (the child process dies) and then return to the top of the list

Different shells

There are many different types of shells. Table 6.1 provides a list of some of the more popular UNIX shells. Under Linux most users will be using `bash`, the **B**ourne **A**gain **Sh**ell. `bash` is an extension of the Bourne shell and uses the Bourne shell syntax. All of the examples in this text are written using the `bash syntax`.

All shells fulfil the same basic responsibilities. The main differences between shells include

- the extra features provided
  Many shells provide command history, command line editing, command completion and other special features.
- the syntax
  Different shells use slightly different syntax for some commands.

Shell	Program name	Description
Bourne shell	`sh`	the original shell from AT&T, available on all UNIX machines
C shell	`csh`	shell developed as part of BSD UNIX
Korn shell	`ksh`	AT&T improvement of the Bourne shell
Bourne again shell	`bash`	Shell distributed with Linux, version of Bourne shell that includes command line editing and other nice things

Table 6.1
Different UNIX shells

The C shell and its various versions have been popular in some fields. However, there are a number of problems with the C shell. The 85321 Website contains a pointer to a document entitled "C Shell Considered Harmful". If you really want to know why we use the Bourne shell syntax read this document.

Starting a shell

When you log onto a UNIX machine the UNIX login process automatically executes a shell for you. Which shell is executed is defined in the last field of your entry in the `/etc/passwd` file.

The last field of every line of `/etc/passwd` specifies which program to execute when the user logs in. The program is usually a shell (but it doesn't have to be).

Exercises

6.1.	What shell is started when you login?

The shell itself is just another executable program. This means you can choose to run another shell in the same way you would run any other command by simply typing in the name of the executable file. When you do the shell you are currently running will find the program and execute it.

To exit a shell any of the following may work (depending on how your environment is set up).

- logout
- exit
- CTRL-D
  By default control D is the end of file (EOF) marker in UNIX. By pressing CTRL-D you are telling the shell that it has reached the end of the file and so it exits. In a later chapter which examines shell programming you will see why shells work with files.

For example

The following is a simple example of starting other shells. Most different shells use a different command-line prompt.

```
bash$ sh
$ csh
% tcsh
> exit
%
$
bash$
```

In the above my original login shell is `bash`. A number of different shells are then started up. Each new shell in this example changes the prompt (this doesn't always happen). After starting up the `tcsh` shell I've then exited out of all the new shells and returned to the original `bash`.

Parsing the command line

The first task the shell performs when you enter a command is to parse the command line. This means the shell takes what you typed in and breaks it up into components and also changes the command-line if certain special characters exist. Special characters are used for a number of purposes and are used to modify the operation of the shell.

Table 6.2 lists most of the special characters which the shell recognises and the meaning the shell places on these characters. In the following discussion the effect of this meaning and what the shell does with these special characters will be explained in more detail.

Character(s)	Meaning
white space	Any white space characters (tabs, spaces) are used to separate arguments multiple white space characters are ignored
newline character	used to indicate the end of the command-line
' " \	special quote characters that change the way the shell interprets special characters
&	Used after a command, tells the shell to run the command in the background
< >> << ' |	I/O redirection characters
* ? [ ] [^	filename substitution characters
$	indicate a shell variable
;	used to separate multiple commands on the one line

Table 6.2
Shell special characters

The Command Line

The following section examines, and attempts to explain, the special shell characters which influence the command line. This influence includes

- breaking the command line into arguments
- allows more than one command to a line
- allows commands to be run in the background

Arguments

One of the first steps for the shell is to break the line of text entered by the user into arguments. This is usually the task of whitespace characters.

What will the following command display?

```
echo hello          there my friend
```

It won't display

```
hello        there my friend
```

instead it will display

```
hello there my friend
```

When the shell examines the text of a command it divides it into the command and a list of arguments. A white space character separates the command and each argument. Any duplicate white space characters are **ignored**. The following diagram demonstrates.

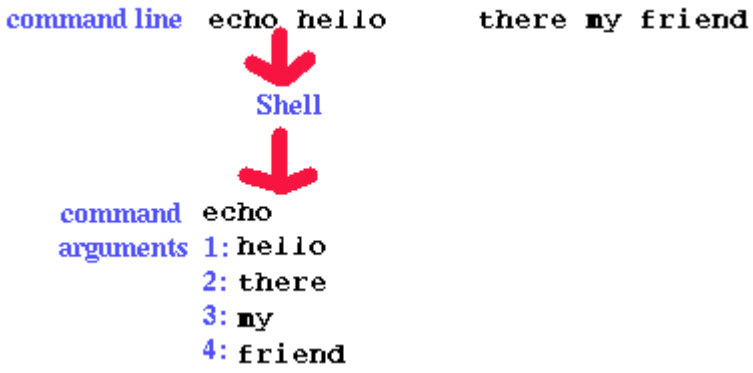


Figure 6.1
Shells, white space and arguments

Eventually the shell will execute the command. The shell passes to the command a list of arguments. The command then proceeds to perform its function. In the case above the command the user entered was the `echo` command. The purpose of the `echo` command is to display each of its arguments onto the screen separated by a single space character.

The important part here is that the `echo` command never sees all the extra space characters between `hello` and `there`. The shell removes this whilst it is performing its parsing of the command line.

One command to a line

The second shell special character in Table 6.2 is the newline character. The newline character tells the shell that the user has finished entering a command and that the shell should start parsing and then executing the command. The shell makes a number of assumptions about the command line a user has entered including

- there is only one command to each line
- the shell should not present the next command prompt until the command the user entered is finished executing.

This section examines how some of the shell special characters can be used to change these assumptions.

Multiple commands to a line

The `;` character can be used to place multiple commands onto the one line.

```
ls ; cd /etc ; ls
```

The shell sees the `;` characters and knows that this indicates the end of one command and the start of another.

Commands in the background

By default the shell will wait until the command it is running for the user has finished executing before presenting the next command line prompt. This

default operation can be changed by using the & character. The & character tells the shell that it should immediately present the next command line prompt and run the command in the background.

This provides major benefits if the command you are executing is going to take a long time to complete. Running it in the background allows you to go on and perform other commands without having to wait for it to complete.

However, you won't wish to use this all the time as some confusion between the output of the command running in the background and shell command prompt can occur.

For example

The `sleep` command usually takes on argument, a number. This number represents the number of seconds the `sleep` command should wait before finishing. Try the following commands on your system to see the difference the & character can make.

```
bash$ sleep 10
bash$ sleep 10 &
```

Filename substitution

In the great majority of situations you will want to use UNIX commands to manipulate files and directories in some way. To make it easier to manipulate large numbers of commands the UNIX shell recognises a number of characters which should be replaced by filenames.

This process is called ether filename substitution or filename globbing.

For example

You have a directory which contains HTML files (an extension of `.html`), GIF files (an extension of `.gif`), JPEG files (an extension `.jpg`) and a range of other files. You wish to find out how big all the HTML files are.

The hard way to do this is to use the `ls –l` command and type in all the filenames.

The simple method is to use the shell special character `*`, which represents any 0 or more characters in a file name

```
ls –l *.html
```

In the above, the shell sees the `*` character and recognises it as a shell special character. The shell knows that it should replace `*.html` with any files that have filenames which match. That is, have 0 or more characters, followed by `.html`

UNIX doesn’t use extensions

MS-DOS and Windows treat a file''s extension as special. UNIX does not do this. Refer to the previous chapter and its discussion of magic numbers.

Table 6.3 lists the other shell special characters which are used in filename substitution.

Character	What it matches
*	0 or more characters
?	1 character
[ ]	matches any one character between the brackets
[^]	matches any one character NOT in the brackets

Table 6.3
Filename substitution special characters

Some examples of filename substitution include

- `cat *`
  `*` will be replaced by the names of all the files and directories in the current directory. The `cat` command will then display the contents of all those files.
- `ls a*bc`
  `a*bc` matches all filenames that start with `a`, end with `bc` and have any characters in between.
- `ls a?bc`
  `a?bc` matches all filenames that start with `a`, end with `bc` and have only ONE character in between.
- `ls [ic]???`
  `[ic]???` matches any filename that starts with either a `i` or `c` followed by any other three letters.
- `ls [^ic]???`
  Same as the previous command but instead of any file that starts with `i` or `c` match any file that DOESN'T start with `i` or `c`.

Exercises

6.2. Given the following files in your current directory:

```
$ ls
feb86 jan12.89
jan19.89 jan26.89
jan5.89 jan85 jan86 jan87
jan88 mar88 memo1 memo10
memo2 memo2.sv
```

What would be the output from the following commands?

```
echo *
echo *[^0-9]
echo m[a-df-z]*
echo [A-Z]*
echo jan*
echo *.*
echo ?????
echo *89
echo jan?? feb?? mar??
echo [fjm][ae][bnr]
```

Removing special meaning

There will be times when you won't want to use the shell special characters as shell special characters. For example, what happens if you really do want to display

```
hello        there my friend
```

How do you do it?

It's for circumstances like this that the shell provides shell special characters called **quotes**. The quote characters `' " \` tell the shell to ignore the meaning of any shell special character.

To display the above you could use the command

```
echo 'hello        there my friend'
```

The first quote character `'` tells the shell to ignore the meaning of any special character between it and the next `'`. In this case it will ignore the meaning of the multiple space characters. So the `echo` command receives one argument instead of four separate arguments. The following diagram demonstrates.

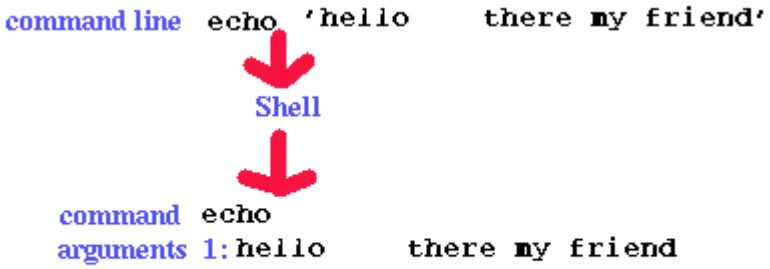


Figure 6.2
Shells, commands and quotes

Table 6.4 lists each of the shell quote characters, their names and how the influence the shell.

Character	Name	Action
'	single quote	the shell will ignore all special characters contained within a pair of single quotes
"	double quote	the shell will ignore all special characters EXCEPT $ ' \ contained within a pair of double quotes
\	backslash	the shell ignores any special character immediately following a backslash

Table 6.4
Quote characters

Examples with quotes

Try the following commands and observe what happens

- `echo I'm David.`
  This causes an error because the ' quote character must be used as one of a pair. Since this line doesn't have a second ' character the shell continues to ignore all the shell special characters it sees, **including** the new line character which indicates the end of a command.
- `echo I\'m David.`
  This is the "correct" implementation of what was attempted above. The \ quote character is used to remove the special meaning of the ' character so it is used as a normal character
- `echo *`
- `echo '*'`

- `echo \*`
  The previous three show two different approaches to removing the special meaning from a single character.
- `echo one two three four`
- `echo 'one two three four'`
- `echo "one two three four"`
- `echo hello there \`
  `my name is david`
  Here the \ is used to ignore the special meaning of the newline character at the end of the first line. This will only work if the newline character is immediately after the \ character. Remember, the \ character only removes the special meaning from the next character.
- `echo files = ; ls`
- `echo files = \; ls`
  Since the special meaning of the `;` character is removed by the \ character means that the shell no longer assumes there are two commands on this line. This means the `ls` characters are treated simply as normal characters, not a command which must be executed.

Exercises

6.3. Create files with the following names
`stars*`
-top
`hello my friend`
`"goodbye"`
Now delete them.

6.4. As was mentioned in the previous chapter the `{}` and `;` used in the `exec` and `ok` actions of the `find` command must be quoted. The normal way of doing this is to use the \ character to remove the special meaning. Why doesn't the use of the single quote character work. e.g. why the following command doesn't work.
`find . -name \*.bak -ok rm '{} ;'`

Input/output redirection

As the name suggests input/output (I/O) redirection is about changing the source of input or destination of output. UNIX I/O redirection is very similar (in part) to MS-DOS I/O redirection (guess who stole from who). I/O redirection, when combined with the UNIX philosophy of writing commands to perform one task, is one of the most important and useful combinations in UNIX.

How it works

All I/O on a UNIX system is achieved using files. This includes I/O to the screen and from a keyboard. Every process under UNIX will open a number of

different files. To keep a track of the files it has, a process maintains a file descriptor for every file it is using.

File descriptors

A file descriptor is a small, non-negative integer. When a process reads/writes to/from a file it passes the kernel the file descriptor and asks it to perform the operation. The kernel knows which file the file descriptor refers to.

Standard file descriptors

Whenever the shell runs a new program (that is when it creates a new process) it automatically opens three file descriptors for the new process. These file descriptors are assigned the numbers `0`, `1` and `2` (numbers from then on are used by file descriptors the process uses). The following table summarises their names, number and default destination.

Name	File descriptor	Default destination
standard input (stdin)	0	the keyboard
standard output (stdout)	1	the screen
standard error (stderr)	2	the screen

Table 6.5
Standard file descriptors

By default whenever a command asks for input it takes that input from standard input. Whenever it produces output it puts that output onto standard output and if the command generates errors then the error messages are placed onto standard error.

For example, the ls command displays an error message when it can't find the file it was given.

```
[root@faile 85321]# ls /fred
ls: /fred: No such file or directory
```

The "No such file or directory" message is sent to standard error.

Changing direction

By using the special characters in the table below it is possible to tell the shell to change the destination for standard input, output and error.

For example

```
cat /etc/passwd > hello
```

tells the shell rather than send the contents of the `/etc/passwd` file to standard output, it should send it to a file called `hello`.

Character(s)	Result
`Command < file`	Take standard input from file
`Command > file`	Place output of `command` into `file`. Overwrite anything already in the file.
`Command >> file`	Append the output of `command` into `file`.
`command << label`	Take standard input for `command` from the following lines until a line that contains `label` by itself
`‘command‘`	execute `command` and replace `‘command‘` with the output of the command
`command1 | command2`	pass the output of `command1` to the input of `command2`
`command1 2> file`	redirect standard error of `command1` to `file`. The `2` can actually be replaced by any number which represents a file descriptor
`command1 >& file_descriptor`	redirect output of `command1` to a `file_descriptor` (the actual number for the file descriptor)

Table 6.6
I/O redirection constructs

Using standard I/O

Not all commands use standard input and standard output. For example the `cd` command doesn't take any input and doesn't produce any output. It simply takes the name of a directory as an argument and changes to that directory. It does however use standard error if it can't change into the directory.

It doesn't make sense to redirect the I/O of some commands

For example, the cp command doesn't produce any output. It may produce errors if it cannot copy the requested file but otherwise there is no output. So

```
cp /etc/passwd /tmp/passwd > output.dat
```

does not make sense.

Filters

On the other hand some commands will always take their input from standard input and put their output onto standard output. All of the filters discussed earlier in the textbook act this way.

As an example lets take the `cat` command mentioned previously. If you execute the `cat` command without supplying it with any parameters it will take its input from standard input and place its output onto standard output.

Try it. Execute the command `cat` with no arguments. Hit `CTRL-D`, on a line by itself,to signal the end of input. You should find that `cat` echoes back to the screen every line you type.

Try the same experiment with the other filters mentioned earlier.

I/O redirection examples

- `ls > the.files`
  Create a file `the.files` that contains the list of files in the current directory.
- `ls /fred 2> /dev/null`
  Send any error messages to the null device (throw it away).
- `cat the.files | more`
  Same effect as the command `more the.files`. Display the content of the file `the.files` one page at a time.
- `ls /etc >> the.files`
  Add the list of files in from the `/etc` directory onto the end of the file `the.files`.
- `echo number of lines in the.files = ‘wc -l the.files‘`
  Execute the command `wc -l the.files`. Replace it with its output and then execute the `echo` command. Will display output similar to `number of lines in the.files = 66`
- `cat << finished > input`
  Ask the user to type in information until they enter a line with just `finished` on it. Then copy all the information entered by the user into the file called `input`
- `cd /etc > output.file`
  Create an empty file called `output.file`. The `cd` command generates no output so redirecting its output creates an empty file.
- `ls | cd`
  An error message. `cd` doesn't accept input so when the shell tries to send the output of the `ls` command to the `cd` command it doesn't work.
- `echo ‘wc -l /etc/passwd‘`
  Execute the `wc` command and pass its output to the `echo` command as arguments.

Redirecting standard error

There will be times where you wish to either throw standard error away, join standard error and standard output, or just view standard error. This section provides examples of how this can be accomplished using I/O redirection.

- the file `xx` doesn't exist display an error message on standard error

```
$ ls xx
/bin/ls: xx: No such file or directory
```

- redirect standard output to the file `errors`, no change

```
$ ls xx > errors
/bin/ls: xx: No such file or directory
```

- redirect standard error to the file `errors` nothing on the screen

```
$ ls xx 2> errors
```

- file `chap1.ps` does exist so we get output but the errors still go to the file

```
$ ls chap1.ps xx 2> errors
chap1.ps
```

- try to send both stdout and stderr to the `errors` file, but stdout doesn't go

```
$ ls chap1.ps xx >& 2 2> errors
chap1.ps
```

- try a different order and it does work, **why?**

```
$ ls chap1.ps xx 2> errors >& 2
```

Evaluating from left to right

The shell evaluates arguments from left to right, that is it works with each argument starting with those from the left. This can influence how you might want to use the I/O redirection special characters.

For example

An example of why this is important is when you want to send both standard output and standard error of a command to the same file.

Lets say we are attempting to view the attributes of the two files `chap1.ps` and `xx`. The idea is that the file `xx` does not exist so the `ls` command will generate an error when it can't find the file. Both the error and the file attributes of the `chap1.ps` file are meant to be sent to a file called `errors`. So we try to use

- `2>&1`
  This should redirect file descriptor 2 to standard output (refer back to Table 6.6). It should make standard error (file descriptor 2) go to the same place as standard output (file descriptor 1)
- `> output.and.errors`
  This sends standard output to the file `output.and.errors` (we hope).

Lets try and find out.

```
$ ls -l chap1.ps xx 2>&1  > output.and.errors
ls: xx: No such file or directory
[david@faile tmp]$ cat output.and.errors
-rw-rw-r--   1 david    david           0 Jan  9 16:23 chap1.ps
```

As you can see it doesn't work. The error message still appears on the screen and doesn't get sent to the output.and.errors file.

Can you explain why?

The reason it doesn't work is that the shell evaluates the arguments of this command from left to right. The order of evaluation goes like this

- `ls`
  The first argument tells the shell what command should be executed.
- `-l`
  The shell won't recognise any special characters in this argument so it will pass it on directly to the command.
- `chap1.ps`
  Again the shell won't see any shell special characters and so passes this argument directly onto the command.
- `xx`
  Same again.
- `2>&1`
  Now some action. The shell recognises some special characters here. It knows that `>&` are I/O redirection characters. These characters tell the shell that it should redirect standard error for this command to the same place as standard output. The current location for standard output is the terminal (the screen). So standard error is redirected to the terminal. **No change from normal.**
- `>`
  Again the shell will see a shell special character. In this case, the shell knows that standard output should be redirected to the location specified in the next argument.
- `output.and.errors`
  This is where the shell will send the standard error of the command, a file called `output.and.errors`.

The outcome of this is that standard output still goes to the terminal and standard error goes to the file `output.and.errors`.

What we wanted is for both standard output and standard error to go to the file. The problem is the order in which the shell evaluated the arguments. The solution is to switch the I/O redirection shell characters.

```
[david@faile tmp]$ ls -l chap1.ps xx > output.and.errors 2>&1
[david@faile tmp]$ cat output.and.errors
ls: xx: No such file or directory
-rw-rw-r--   1 david    david           0 Jan  9 16:23 chap1.ps
```

Changing the order means that standard output is redirected to the file `output.and.errors` FIRST and then standard error is redirected to where standard output is pointing (the same file).

Everything is a file

One of the features of the UNIX operating system is that almost everything can be treated as a file. This combined with I/O redirection allows you to achieve some powerful and interesting results.

You've already seen that by default stdin is the keyboard and stdout is the screen of your terminal. The UNIX operating system treats these devices as files (remember the shell sets up file descriptors for standard input/output). But which file is used?

`tty`

The `tty` command is used to display the filename of the terminal you are using.

```
$ tty
/dev/ttyp1
```

In the above example my terminal is accessed through the file `/dev/ttyp1`. This means if I execute the following command

```
cat /etc/passwd > /dev/ttyp1
```

standard output will be redirected to `/dev/ttyp1` which is where it would've gone anyway.

Exercises

6.5. What would the following command do?
`ls > 'tty'`

Device files

`/dev/ttyp1` is an example of a device file. A device file is a interface to one of the kernel's device drivers. A device driver is a part of the Linux kernel. It knows how to talk to a specific hardware device and presents a standard programming interface that is used by software.

When you redirect I/O to/from a device file the information is passed through the device file, to the device driver and eventually to the hardware device or peripheral. In the previous example the contents of the `/etc/passwd` file were sent through the device file `/dev/ttyp1`, to a device driver. The device driver then displayed it on an appropriate device.

`/dev`

All of the system's device files will be stored under the directory `/dev`. A standard Linux system is likely to have over 600 different device files. The following table summarises some of the device files.

filename	purpose	filename	purpose
/dev/hda	The first IDE disk drive	/dev/hda1	the first partition on the first IDE disk drive
/dev/sda	The first SCSI disk drive	/dev/sda1	the first partition on the first SCSI drive
/dev/audio	Sound card	/dev/cdrom	CD-ROM drive
/dev/fd0	First floppy drive	/dev/ttyS1	the second serial port

Table 6.7
Example device files

Redirecting I/O to device files

As you've seen it is possible to send output or obtain input from a device file. That particular example was fairly boring, here's another.

```
cat beam.au > /dev/audio
```

This one sends a sound file to the audio device. The result (if you have a sound card) is that the sound is played.

When not to

If you examine the file permissions of the device file `/dev/hda1` you'll find that only the `root` user and the group `disk` can write to that file. You should not be able to redirect I/O to/from that device file (unless you are the root user).

If you could it would corrupt the information on the hard-drive. There are other device files that you should not experiment with. These other device file should also be protected with appropriate file permissions.

`/dev/null`

`/dev/null` is the UNIX "garbage bin". Any output redirected to `/dev/null` is thrown away. Any input redirected from `/dev/null` is empty. `/dev/null` can be used to throw away output or create an empty file.

```
cat /etc/passwd > /dev/null
cat > newfile < /dev/null
```

The last command is one way of creating an empty file.

Exercises

6.6. Using I/O redirection how would you perform the following tasks
- display the first field of the `/etc/passwd` file sorted in descending order
- find the number of lines in the `/etc/passwd` file that contain the word `bash`

Shell variables

The shell provides a variable mechanism where you can store information for future use. Shell variables are used for two main purposes: shell programming and environment control. This section provides an introduction to shell variables and their use in environment control. A later chapter discusses shell programming in more detail.

Environment control

Whenever you run a shell it creates an environment. This environment includes pre-defined shell variables used to store special values including

- the format of the prompt the shell will present to you
- your current path
- your home directory
- the type of terminal you are using
- and a great deal more.

Any shell variable you create will be stored within this environment. A later section in this chapter goes into more detail about environment control.

The `set` command

The `set` command can be used to view you shell's environment. By executing the `set` command without any parameters it will display all the shell variables currently within your shell's environment.

Using shell variables

There are two main operations performed with shell variables

- assign a variable a value
- use a variable's value

Assigning a value

Assigning value to a shell variable is much the same as in any programming language *`variable_name=value`*.

```
my_variable=hello
theNum=5
myName="David Jones"
```

A shell variable can be assigned just about any value, though there are a few guidelines to keep in mind.

A space is a shell special character. If you want your shell variable to contain a space you must tell the shell to ignore the space's special meaning. In the above example I've used the double quotes. ***For the same reason there should never be any spaces around the = symbol.***

Accessing a variable's value

To access a shell variable's value we use the `$` symbol. The `$` is a shell special character that indicates to the shell that it should replace a variable with its value.

For example

```
dinbig$ myName="David Jones"
dinbig$ echo My name is $myName
My name is David Jones
dinbig$ command=ls
dinbig$ $command
Mail ethics.txt papers
dinbig$ echo A$empty:
A:
```

Uninitialised variables

The last command in the above example demonstrates what the value of a variable is when you haven't initialised it. The last command tries to access the value for the variable `empty`.

But because the variable `empty` has never been initialised it is totally empty. Notice that the result of the command has nothing between the `A` and the `:`.

Resetting a variable

It is possible to reset the value of a variable as follows

```
myName=
```

This is totally different from trying this

```
myName=' '
```

This example sets the value of `myName` to a space character **NOT** nothing.

The `readonly` command

As you might assume the `readonly` command is used to make a shell variable readonly. Once you execute a command like

```
readonly my_variable
```

The shell variable `my_variable` can no longer be modified.

To get a list of the shell variables that are currently set to read only you run the `readonly` command without any parameters.

The `unset` command

Previously you've been shown that to reset a shell variable to nothing as follows

```
variable=
```

But what happens if you want to remove a shell variable from the current environment? This is where the `unset` command comes in. The command

```
unset variable
```

Will remove a variable completely from the current environment.

There are some restrictions on the `unset` command. You cannot use `unset` on a read only variable or on the pre-defined variables `IFS, PATH, PS1, PS2`

Arithmetic

UNIX shells do not support any notion of numeric data types such as integer or real. All shell variables are strings. How then do you perform arithmetic with shell variables?

One attempt might be

```
dinbig:~$ count=1
dinbig:~$ Rcount=$count+1
```

But it won't work. Think about what happens in the second line. The shell sees `$count` and replaces it with the value of that variable so we get the command `count=1+1`. Since the shell has no notion of an integer data type the variable `count` now takes on the value `1+1` (just a string of characters).

The `expr` command

The UNIX command `expr` is used to evaluate expressions. In particular it can be used to evaluate integer expressions. For example

```
dinbig:~$ expr 5 + 6
11
dinbig:~$ expr 10 / 5
2
dinbig:~$ expr 5 \* 10
50
dinbig:~$ expr 5 + 6 * 10
expr: syntax error
dinbig:~$ expr 5 + 6 \* 10
65
```

Note that the shell special character `*` has to be quoted. If it isn't the shell will replace it with the list of all the files in the current directory which results in `expr` generating a syntax error.

Using `expr`

By combining the `expr` command with the grave character ‘ we have a mechanism for performing arithmetic on shell variables. For example

```
count=1
count=‘expr $count + 1‘
```

`expr` restrictions

The `expr` command only works with integer arithmetic. If you need to perform floating point arithmetic have a look at the `bc` and `awk` commands.

The `expr` command accepts a list of parameters and then attempts to evaluate the expression they form. As with all UNIX commands the parameters for the `expr` command must be separated by spaces. If you don't `expr` interprets the input as a sequence of characters.

```
dinbig:~$ expr 5+6
5+6
dinbig:~$ expr 5+6 \* 10
expr: non-numeric argument
```

Alternatives to expr for arithmetic

The expr command is the traditional approach for perform arithmetic but it is by no means the best and has at least two major draw backs including

- it doesn't handle decimal points
  If you want to add 5.5 and 6.5 you can't do it with expr.
  One solution to this is the bc command

```
[david@faile tmp]$ echo 5.5 + 5 | bc
10.5
```

- every use requires the creation of a new process
  Chapter 5 includes a discussion of why this can be a problem and cause shell scripts to be very slow.
  An alternative to this is to use the arithemetic capabilities provided by many of the modern shells including bash. This is what is used in the add2 script mentioned in the previous chapter.

```
[david@faile tmp]$ echo $[ 5 + 5]
10
```

Valid variable names

Most programming languages have rules that restrict the format of variable names. For the Bourne shell, variable names must

- start with either a letter or an underscore character,
- followed by zero or more letters, numbers or underscores

{}

In some cases you will wish to use the value of a shell variable as part of a larger word. Curly braces `{ }` are used to separate the variable name from the rest of the word.

For example

You want to copy the file `/etc/passwd` into the directory `/home/david`. The following shell variables have been defined.

```
directory=/etc/
home=/home/david
```

A first attempt might be

```
cp $directorypasswd $home
```

This won't work because the shell is looking for the shell variable called `directorypasswd` (there isn't one) instead of the variable `directory`.

The correct solution would be to surround the variable name `directory` with curly braces. This indicates to the shell where the variable stops.

```
cp ${directory}passwd $home
```

Environment control

Whenever you run a shell it creates an environment in which it runs. This environment specifies various things about how the shell looks, feels and operates. To achieve this the shell uses a number of pre-defined shell variables. Table 6.8 summarises these special shell variables.

Variable name	Purpose
HOME	your home directory
SHELL	the executable program for the shell you are using
UID	your user id
USER	your username
TERM	the type of terminal you are using
DISPLAY	your X-Windows display
PATH	your executable path

Table 6.8
Environment variables

PS1 and PS2

The shell variables `PS1` and `PS2` are used to store the value of your command prompt. Changing the values of `PS1` and `PS2` will change what your command prompt looks like.

```
dinbig:~$ echo :$PS1: and :$PS2:
:\h:\w\$ : and :> :
```

`PS2` is the secondary command prompt. It is used when a single command is spread over multiple lines. You can change the values of `PS1` and `PS2` just like you can any other shell variable.

bash extensions

You'll notice that the value of `PS1` above is `\h:\w\$` but my command prompt looks like `dinbig:~$`.

This is because the `bash` shell provides a number of extra facilities. One of those facilities is that it allows the command prompt to contain the hostname `\h`(the name of my machine) and the current working directory `\w`.

With older shells it was not possible to get the command prompt to display the current working directory.

Exercises

6.7.	Many first time users of older shells attempt to get the command prompt to contain the current directory by trying this `PS1='pwd'` The `pwd` command displays the current working directory. Explain why this will not work. (HINT: When is the `pwd` command executed?)

Variables and sub-shells

Every time you start a new shell, the new shell will create a new environment separate from its parent's environment. The new shell will not be able to access or modify the environment of its parent shell.

For example

Here's a simple example.

`dinbig:~$ myName=david`	create a shell variable
`dinbig:~$ echo $myName` `david`	use it
`dinbig:~$ bash`	start a new shell
`dinbig:~$ echo my name is $myName` `my name is`	try to use the parent shell's variable
`dinbig:~$ exit`	exit from the new shell and return to the parent
`dinbig:~$ echo $myName` `david`	use the variable again

As you can see a new shell cannot access or modify the shell variables of its parent shells.

`export`

There are times when you may wish a child or sub-shell to know about a shell variable from the parent shell. For this purpose you use the `export` command. For example,

```
dinbig:~$ myName=David Jones
dinbig:~$ bash
dinbig:~$ echo my name is $myName
my name is
dinbig:~$ logout
dinbig:~$ export myName
dinbig:~$ bash
dinbig:~$ echo my name is $myName
my name is david
dinbig:~$ exit
```

Local variables

When you export a variable to a child shell the child shell creates a local copy of the variable. Any modification to this local variable cannot be seen by the parent process.

There is **no** way in which a child shell can modify a shell variable of a parent process. The `export` command only passes shell variables to child shells. It cannot be used to pass a shell variable from a child shell back to the parent.

For example

```
dinbig:~$ echo my name is $myName
my name is david
dinbig:~$ export myName
dinbig:~$ bash
dinbig:~$ myName=fred     # child shell modifies variable
dinbig:~$ exit
dinbig:~$ echo my name is $myName
my name is david
# there is no change in the parent
```

Advanced variable substitution

The shell provides a number of additional more complex constructs associated with variable substitution. The following table summarises them.

Construct	Purpose
${*variable*:-*value*}	replace this construct with the variable's value if it has one, if it doesn't, use *value* but don't make *variable* equal to *value*
${*variable*:=*value*}	same as the above but if *variable* has no value assign it *value*
${*variable*:?*message*}	replace the construct with the value of the variable if it has one, if it doesn't then display *message* onto stderr if *message* is null then display *prog*: *variable*: parameter null or not set on stderr
${*variable*:+*value*}	if *variable* has a value replace it with *value* otherwise do nothing

Table 6.9
Advanced variable substitution

For example

```
dinbig:~$ myName=
dinbig:~$ echo my name is $myName
my name is
dinbig:~$ echo my name is ${myName:-"NO NAME"}
my name is NO NAME
dinbig:~$ echo my name is $myName
my name is
dinbig:~$ echo my name is ${myName:="NO NAME"}
my name is NO NAME
dinbig:~$ echo my name is $myName
```

```
my name is NO NAME
dinbig:~$ herName=
dinbig:~$ echo her name is ${herName:?"she hasn't got a name"}
bash: herName: she hasn't got a name
dinbig:~$ echo her name is ${herName:?}
bash: herName: parameter null or not set
```

Evaluation order

In this chapter we've looked at the steps the shell performs between getting the user's input and executing the command. The steps include

- I/O redirection
  Where the shell changes the direction in which I/O is being sent.
- variable substitution
  The shell replaces shell variables with the corresponding values.
- filename substitution
  This is where the shell replaces globbing characters with matching filenames.

An important question is in what order does the shell perform these steps?

Why order is important

Look at the following example

```
dinbig:~$ pipe=\|
dinbig:~$ echo $pipe
|
dinbig:~$ star=\*
dinbig:~$ echo $star
Mail News README VMSpec.ps.bak acm.bhx acm2.dot
```

In the case of the `echo $start` command the shell has seen `$star` and replaced it with its value `*`. The shell sees the `*` and replaces it with the list of the files in the current directory.

In the case of the `echo $pipe` command the shell sees `$pipe` and replaces it with its value `|`. It then displays `|` onto the screen.

Why didn't it treat the `|` as a special character? If it had then `echo |` would've produced something like the following.

```
[david@faile tmp]$ echo |
>
```

The >, produced by the shell not typed in by the user, indicates that the shell is still waiting for input. The shell is still expecting another command name.

The reason this isn't produced in the previous example is related to the order in which the shell performs its analysis of shell special variables.

The order

The order in which the shell performs the steps is

- I/O redirection

- variable substitution
- filename substitution

For the command

```
echo $PIPE
```

the shell performs the following steps

- check for any I/O redirection characters, there aren't any, the command line is currently `echo $PIPE`
- check for variables, there is one `$PIPE`, replace it with its value, the command line is now `echo |`
- check for any wildcards, there aren't any

So it now executes the command `echo |`.

If you do the same walk through for the `echo $star` command you should see how its output is achieved.

The `eval` command

What happens if I want to execute the following command

```
ls $pipe more
```

using the shell variable `pipe` from the example above?

The intention is that the `pipe` shell variable should be replaced by its value `|` and that the `|` be used to redirect the output of the `ls` command to the `more` command.

Due to the order in which the shell performs its evaluation this won't work.

Doing it twice

The `eval` command is used to evaluate the command line twice. `eval` is a built-in shell command. Take the following command (using the `pipe` shell variable from above)

```
eval ls $pipe more
```

The shell sees the `$pipe` and replaces it with its value, `|`. It then executes the `eval` command.

The `eval` command repeats the shell's analysis of its arguments. In this case it will see the `|` and perform necessary I/O redirection while running the commands.

Conclusion

The UNIX command line interface is provided by programs called shells. A shell's responsibilities include

- providing the command line interface
- performing I/O redirection

- performing filename substitution
- performing variable substitution
- and providing an interpreted programming language

A shell recognises a number of characters as having special meaning. Whenever it sees these special characters it performs a number of tasks that replace the special characters.

When a shell is executed it creates an environment in which to run. This environment consists of all the shell variables created including a number of pre-defined shell variables that control its operation and appearance.

Review Questions

6.1

What is the effect of the following command sequences?

- `ls | wc -l`
- `rm ???`
- `who | wc -l`
- `mv progs/* /usr/steve/backup`
- `ls *.c | wc -l`
- `rm *.o`
- `who | sort`
- `cd ; pwd`
- `cp memo1 ..`
- `ls -l | sort +4n`

6.2

What is the output of the following commands? Are there any problems? How would you fix it?

- `echo this is a star *`
- `echo ain\\\\’t you my friend`
- `echo "** hello **"`
- `echo "the output of the ls command is ‘ls‘"`
- `echo ‘the output of the pwd command is ‘pwd‘‘`

6.3

Which of the following are valid shell variable names?

- `XxXxXxXx`
- `_`
- `12345`

- `HOMEDIR`
- `file.name`
- `_date`
- `file_name`
- `x0-9`
- `file1`
- `Slimit`

6.4

Suppose your HOME directory is `/usr/steve` and that you have sub-directory as shown in figure 6.3.

Assuming you just logged onto the system and executed the following commands:

```
docs=/usr/steve/documents
let=$docs/letters
prop=$docs/proposals
```

write commands to do the following using these variables

- List the contents of the `documents` directory
- Copy all files from the `letters` directory to the `proposals` directory
- Move all files with names that contain a capital letter from the `letters` directory to the current directory.
- Count the number of files in the `memos` directory.

What would be the effect of the following commands?

- `ls $let/..`
- `cat $prop/sys.A >> $let/no.JSK`
- `echo $let/*`
- `cp $let/no.JSK $prop`
- `cd $prop`
- `files_in_prop=‘echo $prop*‘`
- `cat ‘echo $let\*‘`

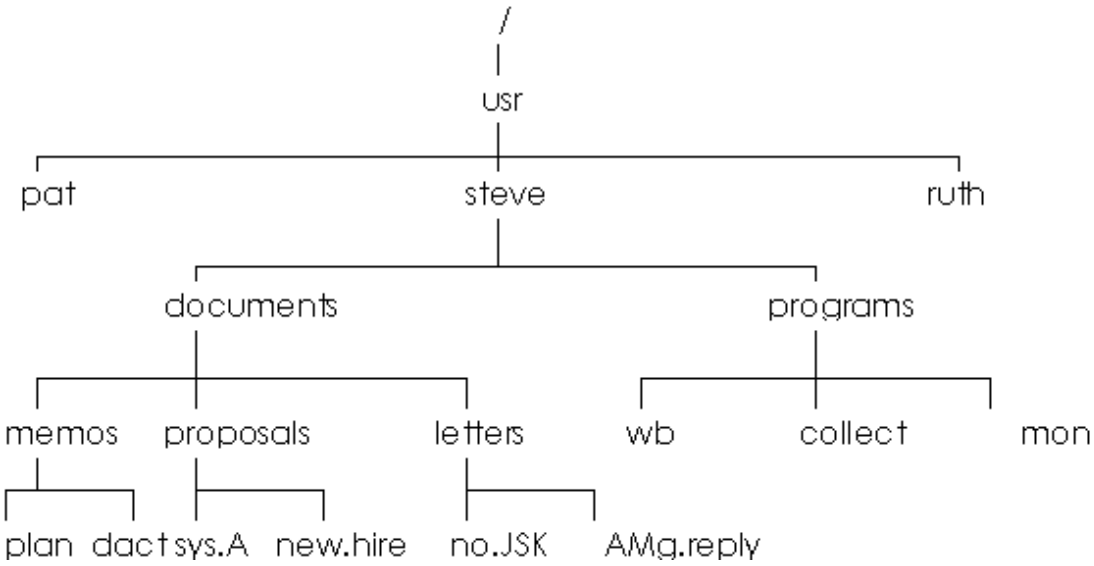


Figure 6.3
Review Question 6.4

Chapter 7

Text Manipulation

Introduction

Many of the tasks a Systems Administrator will perform involve the manipulation of textual information. Some examples include manipulating system log files to generate reports and modifying shell programs. Manipulating textual information is something which UNIX is quite good at and provides a number of tools which make tasks like this quite simple, once you understand how to use the tools. The aim of this chapter is to provide you with an understanding of these tools

By the end of this chapter you should be

- familiar with using regular expressions,
- able to use regular expressions and `ex` commands to perform powerful text manipulation tasks.
- Online lecture 7 on the 85321 Website/CD-ROM provides some alternative discussion of the topics covered in this chapter. It may be beneficial to follow that lecture in conjunction with reading this chapter.

Regular expressions

Regular expressions provide a powerful method for matching patterns of characters. Regular expressions (REs) are understood by a number of commands including `ed ex sed awk grep egrep, expr` and even `vi`.

Some examples of regular expressions look like include

- `david`
  Will match any occurrence of the word `david`
- `[Dd]avid`
  Will match either `david` or `David`
- `.avid`
  Will match any letter (`.`) followed by `avid`
- `^david$`
  Will match any line that contains only `david`
- `d*avid`
  Will match `avid`, `david`, `ddavid` `dddavid` and any other word with repeated `d`s followed by `avid`
- `^[^abcef]avid$`
  Will match any line with only five characters on the line, where the last

four characters must be `avid` and the first character can be any character except `abcef`.

Each regular expression is a pattern, it matches a collection of characters. That means by itself the regular expression can do nothing. It has to be combined with some UNIX commands which understand regular expressions. The simplest example of how regular expressions are used by commands is the `grep` command.

The `grep` command was introduced in a previous chapter and is used to search through a file and find lines that contain particular patterns of characters. Once it finds such a line, by default, the `grep` command will display that line onto standard output. In that previous chapter you were told that `grep` stood for global regular expression pattern match. Hopefully you now have some idea of where the regular expression part comes in.

This means that the patterns that `grep` searches for are regular expressions.

The following are some example command lines making use of the `grep` command and regular expressions

- `grep unix tmp.doc`
  find any lines contain `unix`
- `grep '[Uu]nix' tmp.doc`
  find any lines containing either `unix` or `Unix`. ***Notice that the regular expression must be quoted.*** This is to prevent the shell from treating the `[]` as shell special characters and performing file name substitution.
- `grep '[^aeiouAEIOU]*' tmp.doc`
  Match any number of characters that do not contain a vowel.
- `grep '^abc$' tmp.doc`
  Match any line that contains only `abc`.
- `grep 'hel.' tmp.doc`
  Match `hel` followed by any other character.

Other UNIX commands which use regular expressions include sed, ex and vi. Which are editors (different types of editors) which allow the use of regular expressions to search and to search and replace patterns of characters. Much of the power of the Perl script language and the awk command can also be traced back to regular expressions.

You will also find that the use of regular expressions on other platforms (i.e. Microsoft) is increasing as the benefits of REs become apparent.

REs versus filename substitution

It is important that you realise that regular expressions are different from filename substitution. If you look in the previous examples using `grep` you will see that the regular expressions are sometimes quoted. One example of this is the command

```
grep '[^aeiouAEIOU]*' tmp.doc
```

Remember that [^] and * are all shell special characters. If the quote characters (`''`) were not there the shell would perform filename substitution and replace these special characters with matching filenames.

For example, if I execute the above command without the quote characters in one of the directories on my Linux machine the following happens

```
[david@faile tmp]$ grep [^aeiouAEIOU]* tmp.doc
tmp.doc:chap1.ps this is the line to match
```

The output here indicates that grep found one line in the file tmp.doc which contained the regular expression pattern it wanted and it has displayed that line. However this output is wrong.

Remember, before the command is executed the shell will look for and modify any shell special characters it can find. In this command line the shell will find the regular expression because it contains special characters. It replaces the [^aeiouAEIOU]* with all the files in the current directory which don't start with a vowel (aeiouAEIOU).

The following sequence shows what is going on. First the ls command is used to find out what files are in the current directory. The echo command is then used to discover which filenames will be matched by the regular expression. You will notice how the file anna is not selected (it starts with an a).

The grep command then shows how when you replace the attempted regular expression with what the shell will do you get the same output as the grep command above with the regular expression.

```
[david@faile tmp]$ ls
anna  chap1.ps    magic  tmp  tmp.doc
[david@faile tmp]$ echo [^aeiouAEIOU]*
chap1.ps magic tmp tmp.doc
[david@faile tmp]$ grep chap1.ps magic tmp tmp.doc
tmp.doc:chap1.ps this is the line to match
```

In this example command we do not want this to happen. We want the shell to ignore these special characters and pass them to the `grep` command. The grep command understands regular expressions and will treat them as such. The output of the proper command on my system is

```
[david@faile tmp]$ grep '[^aeiouAEIOU]*' tmp.doc
This is atest
chap1.ps this is the line to match
```

Regular expressions have nothing to do with filename substitution, they are in fact completely different. Table 7.1 highlights the differences between regular expressions and filename substitution.

Filename substitution	Regular expressions
Performed by the shell	Performed by individual commands
used to match filenames	Used to match patterns of characters in data files

Table 7.1
Regular expressions versus filename substitution

How they work

Regular expressions use a number of special characters to match patterns of characters. Table 7.2 outlines these special characters and the patterns they match.

Character	Matches
c	if *c* is any character other than \ [. * ^] $ then it will match a single occurrence of that character
\	remove the special meaning from the following character
.	any one character
^	the start of a line
$	the end of a line
*	0 or more matches of the previous RE
[*chars*]	any one character in *chars* a list of characters
[^*chars*]	any one character NOT in *chars* a list of characters

Table 7.2
Regular expression characters

Exercises

7.4. What will the following simple regular expressions match?

```
fred
[^D]aily
..^end$
he..o
he\.\.o
\$fred
$fred
```

Repetition

There are times when you wish to repeat a previous regular expression. For example, I want to match 40 letter a's. One approach would be to write

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Another would be to use one of the repetition characters from Table 7.3.

```
a\{40,40\}
```

Construct	Purpose
+	match one or more occurrences of the previous RE
?	match zero or one occurrences of the previous RE
\{*n*\}	match exactly *n* occurrences of the previous RE
\{*n*,\}	match at least *n* occurrences of the previous RE
\{*n*, *m*\}	match between *n* and *m* occurrences of the previous RE

Table 7.3
Regular expression repetition characters

Each of the repetition characters in the previous table will repeat the previous regular expression. For example,

- `d+`
  Match one or more d's
- `fred?`
  Match fre followed by 0 or more d's. NOT 0 or more repetitions of fred.
- `.\{5,\}`
  Does not match 5 or more repeats of the same character (e.g. aaaaa). Instead it matches at least 5 or more repeats of any single character.

This last example is an important one. The repetition characters match the previous regular expression and NOT what the regular expression matches. The following commands show the distinction

```
[david@faile tmp]$ cat pattern
aaaaaaaaaaa
david
dawn
[david@faile tmp]$ grep '.\{5,\}' pattern
aaaaaaaaaaa
david
```

First step is to show the contents of the file pattern, three lines of text, one with a row of a's, another with the name david and another with the name dawn. If the regular expression `.\{5,\}` is meant to match at least 5 occurrences of the same character it should only match the line with all a's. However, as you can see it also matches the line containing david.

The reason for this is that `.\{5,\}` will match any line with at least 5 single characters. So it does mention the line with the name david but doesn't match the line with the name dawn. That last line isn't matched because it only contains 4 characters.

Concatenation and Alternation

It is quite common to concatenate regular expressions one after the other. In this situation any string which the regular expression matches will match the entire regular expression. Alternation, choosing between two or regular expressions is done using the | character. For example,

- `egrep '(a|b)' pattern`
  Match any line that contains either an `a` or a `b`

Different commands, different REs

Regular expressions are one area in which the heterogeneous nature of UNIX becomes apparent. Different programs on different platforms recognise different subsets of regular expressions. You need to refer to the manual page of the various commands to find out which features it supports. On Linux you can also check the regex(7) manual page for more details about the POSIX 1003.2 regular expressions supported by most of the GNU commands used by Linux.

One example of the difference, using the pattern file used above, follows

```
[david@faile tmp]$ grep '.\{2,\}' pattern
aaaaaaaaaaa
david
[david@faile tmp]$ egrep '.\{2,\}' pattern
```

This demonstrates how the grep and egrep commands on Linux use slightly different versions of regular expressions.

Exercises

7.5. Write grep commands that use REs to carry out the following.

1. Find any line starting with `j` in the file `/etc/passwd` (equivalent to asking
   to find any username that starts with j).
2. Find any user that has a username that starts with `j` and uses `bash` as their login shell (if they use `bash` their entry in `/etc/passwd` will end with
   the full path for the `bash` program).
3. Find any user that belongs to a group with a group ID between 0 and 99
   (group id is the fourth field on each line in `/etc/passwd`).

Tagging

Tagging is an extension to regular expressions which allows you to recognise a particular pattern and store it away for future use. For example, consider the regular expression

```
da\(vid\)
```

The portion of the RE surrounded by the `\(` and `\)` is being tagged. Any pattern of characters that matches the tagged RE, in this case `vid`, will be stored in a register. The commands that support tagging provide a number of registers in which character patterns can be stored.

It is possible to use the contents of a register in a RE. For example,

```
\(abc\)\1\1
```

The first part of this RE defines the pattern that will be tagged and placed into the first register (remember this pattern can be any regular expression). In this case the first register will contain `abc`. The 2 following `\1` will be replaced by the contents of register number 1. So this particular example will match `abcabcabc`.

The `\` characters must be used to remove the other meaning which the brackets and numbers have in a regular expression.

For example

Some example REs using tagging include

- `\(david\)\1`
  This RE will match `daviddavid`. It first matches `david` and stores it into the first register (`\(david\)`). It then matches the contents of the first register (`\1`).
- `\(.\)oo\1`
  Will match words such as `noon`, `moom`.

For the remaining RE examples and exercises I'll be referring to a file called `pattern`. The following is the contents of `pattern`.

```
a
hellohello
goodbye
friend how hello
there how are you how are you
ab
bb
aaa
lll
Parameters
param
```

Exercises

7.6. What will the following commands do

```
grep '\(a\)\1' pattern
grep '\(.*\)\1' pattern
grep '\( .*\)\1' pattern
```

ex, ed, sed and vi

So far you''ve been introduced to what regular expressions do and how they work. In this section you will be introduced to some of the commands which allow you to use regular expressions to achieve some quite powerful results.

In the days of yore UNIX did not have full screen editors. Instead the users of the day used the line editor `ed`. `ed` was the first UNIX editor and its impact can be seen in commands such as `sed`, `awk`, `grep` and a collection of editors including `ex` and `vi`.

`vi` was written by Bill Joy while he was a graduate student at the University of California at Berkeley (a University responsible for many UNIX innovations). Bill went on to do other things including being involved in the creation of Sun Microsystems.

`vi` is actually a full-screen version of `ex`. Whenever you use `:wq` to save and quit out of `vi` you are using a `ex` command.

So???

All very exciting stuff but what does it mean to you a trainee Systems Administrator? It actually has at least three major impacts

- by using `vi` you can become familiar with the `ed` commands
- `ed` commands allow you to use regular expressions to manipulate and modify text
- those same `ed` commands, with regular expressions, can be used with `sed` to perform all these tasks non-interactively (this means they can be automated).

Why use `ed`?

Why would anyone ever want to use a line editor like `ed`?

Well in some instances the Systems Administrator doesn't have a choice. There are circumstances where you will not be able to use a full screen editor like `vi`. In these situations a line editor like `ed` or `ex` will be your only option.

One example of this is when you boot a Linux machine with installation boot and root disks. A few years ago these disks usually didn't have space for a full screen editor but they did have `ed`.

`ed` commands

`ed` is a line editor that recognises a number of commands that can manipulate text. Both `vi` and `sed` recognise these same commands. In `vi` whenever you use the `:` command you are using `ed` commands. `ed` commands use the following format.

```
[ address [, address]] command [parameters]
```

(you should be aware that anything between `[]` is optional)

This means that every `ed` command consists of

- 0 or more addresses that specify which lines the command should be performed upon,
- a single character command, and
- an optional parameter (depending on the command)

For example

Some example ed commands include

- `1,$s/old/new/g`
  The address is `1,$` which specifies all lines. The command is the substitute command. With the following text forming the parameters to the command. This particular command will substitute all occurrences of the work `old` with the word `new` for all lines within the current file.
- `4d3`
  The address is line `4`. The command is `d`elete. The parameter `3` specifies how many lines to delete. This command will delete 3 lines starting from line 4.
- `d`
  Same command, `d`elete but no address or parameters. The default address is the current line and the default number of lines to delete is one. So this command deletes the current line.
- `1,10w/tmp/hello`
  The address is from line `1` to line `10`. The command is `w`rite to file. This command will write lines 1 to 10 into the file `/tmp/hello`

The current line

The ed family of editors keep track of the current line. By default any ed command is performed on the current line. Using the address mechanism it is possible to specify another line or a range of lines on which the command should be performed.

Table 7.4 summarises the possible formats for ed addresses.

Address	Purpose
`.`	the current line
`$`	the last line
`7`	line 7, any number matches that line number
`a`	the line that has been marked as `a`
/RE/	the next line matching the *`RE`* moving forward from the current line
?RE?	the next line matching the *`RE`* moving backward from the current line
Address+n	the line that is *`n`* lines after the line specified by *`address`*
Address-n	the line that is *`n`* lines before the line specified by *`address`*
Address1, address2	a range of lines from *`address1`* to *`address2`*
`,`	the same as `1,$`, i.e. the entire file from line `1` to the last line (`$`)
`;`	the same as `.,$`, i.e. from the current line (`.`) to the last line (`$`)

Table 7.4
`ed` addresses

`ed` commands

Regular users of `vi` will be familiar with the `ed` commands `w` and `q` (write and quit). `ed` also recognises commands to delete lines of text, to replace characters with other characters and a number of other functions.

Table 7.5 summarises some of the `ed` commands and their formats. In Table 7.5 *`range`* can match any of the address formats outlined in Table 7.4.

Address	Purpose
*line*a	the append command, allows the user to add text after line number *line*
range d *buffer count*	the delete command, delete the lines specified by *range* and *count* and place them into the buffer *buffer*
range j *count*	the join command, takes the lines specified by *range* and *count* and makes them one line
q	quit
line r *file*	the read command, read the contents of the file *file* and place them after the line *line*
sh	start up a new shell
range s/*RE*/*characters*/*options*	the substitute command, find any characters that match *RE* and replace them with *characters* but only in the range specified by *range*
u	the undo command,
range w *file*	the write command, write to the file *file* all the lines specified by *range*

Table 7.5
ed commands

For example

Some more examples of ed commands include

- 5,10s/hello/HELLO/
  replace the **first** occurrence of hello with HELLO for all lines between 5 *and* 10
- 5,10s/hello/HELLO/g
  replace **all** occurrences of hello with HELLO for all lines between 5 and 10
- 1,$s/^\(.\{20,20\}\)\(.*\)$/\2\1/
  for all lines in the file, take the first 20 characters and put them at the end of the line

The last example

The last example deserves a bit more explanation. Let's break it down into its components

- `1,$s`
  The `1,$` is the range for the command. In this case it is the whole file (from line 1 to the last line). The command is substitute so we are going to replace some text with some other text.
- `/^`
  The `/` indicates the start of the RE. The `^` is a RE pattern and it is used to match the start of a line (see Table 7.2).
- `\(.\{20,20\}\)`
  This RE fragment `.\{20,20\}` will match any 20 characters. By surrounding it with `\( \)` those 20 characters will be stored in register 1.
- `\(.*\)$`
  The `.*` says match any number of characters and surrounding it with `\( \)` means those characters will be placed into the next available register (register 2). The `$` is the RE character that matches the end of the line. So this fragment takes all the characters after the first 20 until the end of the line and places them into register 2.
- `/\2\1/`
  This specifies what text should replace the characters matched by the previous RE. In this case the `\2` and the `\1` refer to registers 1 and 2. Remember from above that the first 20 characters on the line have been placed into register 1 and the remainder of the line into register 2.

The `sed` command

`sed` is a non-interactive version of `ed`. `sed` is given a sequence of `ed` commands and then performs those commands on its standard input or on files passes as parameters. It is an extremely useful tool for a Systems Administrator. The `ed` and `vi` commands are interactive which means they require a human being to perform the tasks. On the other had `sed` is non-interactive and can be used in shell programs which means tasks can be automated.

`sed` command format

By default the `sed` command acts like a filter. It takes input from standard input and places output onto standard output. `sed` can be run using a number of different formats.

```
sed command [file-list]
sed [-e command] [-f command_file] [filelist]
```

`command` is one of the valid `ed` commands.

The `-e` *`command`* option can be used to specify multiple `sed` commands. For example,

```
sed –e '1,$s/david/DAVID/' –e '1,$s/bash/BASH/' /etc/passwd
```

The `-f command_file` tells `sed` to take its commands from the file `command_file`. That file will contain `ed` commands one to a line.

For example

Some of the tasks you might use `sed` for include

- change the username `DAVID` in the `/etc/passwd` to `david`
- for any users that are currently using `bash` as their login shell change them over to the `csh`.

You could also use `vi` or `ed` to perform these same tasks. Note how the `/` in `/bin/bash` and `/bin/csh` have been quoted. This is because the `/` character is used by the substitute command to split the text to find and the text to replace it with. It is necessary to quote the `/` character so `ed` will treat it as a normal character.

```
sed 's/DAVID/david/' /etc/passwd
sed 's/david/DAVID/' -e 's/\/bin\/bash/\/bin\/csh/' /etc/passwd
sed -f commands /etc/passwd
```

The last example assumes that there is a file called `commands` that contains the following

```
s/david/DAVID/
s/\/bin\/bash/\/bin\/csh/
```

Understanding complex commands

When you combine regular expressions with ed commands you can get quite a long string of nearly incomprehensible characters. This can be quite difficult especially when you are just starting out with regular expressions. The secret to understanding these strings, like with many other difficult tasks, is breaking it down into smaller components.

In particular, you need to learn to read the regular expression from the left to the right and understand each character as you go.

For example, lets take the second substitute command from the last section

```
s/\/bin\/bash/\/bin\/csh/
```

We know it is an ed command so the first few characters are going to indicate what type of command. Going through the characters

- `s`
  The first character is an s followed by a / so that indicates a substitute command.
  Trouble is we don't know what the range is because it isn't specified. For most commands there will be a default value for the range. For the case of sed the default range is the current line.
- `/`
  In this position it indicates the start of the string the substitute command will search for.

- `\`
  We are now in the RE specifying the string to match. The \ is going to remove the special meaning from the next character.
- `/`
  Normally this would indicate the end of the string to match. However, the previous character has removed that special meaning. Instead we now know the first character we are matching a /
- `bin`
  I've placed these together as they are normal characters. We are now trying to match /bin
- `\/`
  As before the \ removes the special meaning. So we are trying to match /bin/
- `bash`
  Now matching /bin/bash
- `/`
  No \ to remove the special meaning. So this indicates the end of the string to search for and the start of the replace string.

Hopefully you have the idea by now and complete this process. This command will search for the string /bin/bash and replace it with /bin/csh

Exercises

7.7. Perform the following tasks with both `vi` and `sed`.
You have just written a history of the UNIX operating system but you referred to UNIX as `unix` throughout. Replace all occurrences of `unix` with `UNIX`
You've just written a Pascal procedure using `Write` instead of `Writeln`. The procedure is part of a larger program. Replace `Write` with `Writeln` for all lines between the next occurrence of BEGIN and the following END
When you forward a mail message using the `elm` mail program it automatically adds `>` to the beginning of every line. Delete all occurrences of `>` that start a line.

7.8. What do the following `ed` commands do?

```
.+1,$d
1,$s/OSF/Open Software Foundation/g
1,/end/s/\([a-z]*\) \([0-9]*\)/\2 \1/
```

7.9. What are the following commands trying to do? Will they work? If not why not?

```
sed –e 1,$s/^:/fred:/g /etc/passwd
sed '1,$s/david/DAVID/' '1,$s/bash/BASH/' /etc/passwd
```

Conclusions

Regular expressions (REs) are a powerful mechanism for matching patterns of characters. REs are understood by a number of commands including `vi`, `grep`, `sed`, `ed`, `awk` and `Perl`.

`vi` is just one of a family of editors starting with `ed` and including `ex` and `sed`. This entire family recognise `ed` commands that support the use of regular expressions to manipulate text.

Review Questions

7.1

Use `vi` and `awk` to perform the following tasks with the file 85321.txt (the student numbers have been changed to protect the innocent). This file is available from the 85321 Web site/CD-ROM under the resource materials section for week 3. Unless specified assume each task starts with the original file.

- remove the student number
- switch the order for first name, last name
- remove any student with the name david

Chapter 8
Shell Programming

Introduction

Shell Programming - WHY?

While it is very nice to have a shell at which you can issue commands, have you had the feeling that something is missing? Do you feel the urge to issue multiple commands by only typing one word? Do you feel the need for variables, logic conditions and loops? Do you strive for automation?

If so, then welcome to shell programming.

(If you answered no to any of the above then you are obviously in the wrong frame of mind to be reading this - please try again later :)

Shell programming allows system administrators (and users) to create small (and occasionally not-so-small) programs for various purposes including automation of system administration tasks, text processing and installation of software.

Perhaps the most important reason why a Systems Administrator needs to be able to read and understand shell scripts is the UNIX startup process. UNIX uses a large number of shell scripts to perform a lot of necessary system configuration when the computer first starts. If you can't read shell scripts you can't modify or fix the startup process.

Shell Programming - WHAT?

A shell program (sometimes referred to as a shell script) is a text file containing shell and UNIX commands. Remember - a UNIX command is a physical program (like **`cat, cut and grep`**) where as a shell command is an "interpreted" command - there isn''t a physical file associated with the command; when the shell sees the command, the shell itself performs certain actions (for example, **`echo`**)

When a shell program is executed the shell reads the contents of the file line by line. Each line is executed as if you were typing it at the shell prompt. There isn't anything that you can place in a shell program that you can't type at the shell prompt.

Shell programs contain most things you would expect to find in a simple programming language. Programs can contain services including:

- variables
- logic constructs (IF THEN AND OR etc)
- looping constructs (WHILE FOR)

- functions
- comments (strangely the most least used service)

The way in which these services are implemented is dependant on the shell that is being used (remember - there is more than one shell). While the variations are often not major it does mean that a program written for the bourne shell (`sh`/`bash`) will not run in the c shell (`csh`). All the examples in this chapter are written for the bourne shell.

Shell Programming - HOW?

Shell programs are a little different from what you'd usually class as a program. They are plain text and they don't need to be compiled. The shell "interprets" shell programs - the shell reads the shell program line by line and executes the commands it encounters. If it encounters an error (syntax or execution), it is just as if you typed the command at the shell prompt - an error is displayed.

This is in contrast to C/C++, Pascal and Ada programs (to name but a few) which have source in plain text, but require compiling and linking to produce a final executable program.

So, what are the real differences between the two types of programs? At the most basic level, interpreted programs are typically quick to write/modify and execute (generally in that order and in a seemingly endless loop :). Compiled programs typically require writing, compiling, linking and executing, thus are generally more time consuming to develop and test.

However, when it comes to executing the finished programs, the execution speeds are often widely separated. A compiled/linked program is a binary file containing a collection direct systems calls. The interpreted program, on the other hand, must first be processed by the shell which then converts the commands to system calls or calls other binaries - this makes shell programs slow in comparison. In other words, shell programs are not generally efficient on CPU time.

Is there a happy medium? Yes! It is called Perl. Perl is an interpreted language but is interpreted by an extremely fast, optimised interpreter. It is worth noting that a Perl program will be executed inside one process, whereas a shell program will be interpreted from a parent process but may launch many child processes in the form of UNIX commands (ie. each call to a UNIX command is executed in a new process). However, Perl is a far more difficult (but extremely powerful) tool to learn - and this chapter is called "Shell Programming"...

The Basics

A Basic Program

It is traditional at this stage to write the standard "Hello World" program. To do this in a shell program is so obscenely easy that we're going to examine

something a bit more complex - a hello world program that knows who you are...

To create your shell program, you must first edit a file - name it something like "hello", "hello world" or something equally as imaginative - just don't call it "test" - we will explain why later.

In the editor, type the following (or you could go to the 85321 website/CD-ROM and cut and paste the text from the appropriate web page)

```
#!/bin/bash
# This is a program that says hello
echo "Hello $LOGNAME, I hope you have a nice day!"
```

(You may change the text of line three to reflect your current mood if you wish)

Now, at the prompt, type the name of your program - you should see something like:

```
bash: ./helloworld: Permission denied
```

Why?

The reason is that your shell program isn't executable because it doesn't have its execution permissions set. After setting these (Hint: something involving the `chmod` command), you may execute the program by again typing its name at the prompt.

An alternate way of executing shell programs is to issue a command at the shell prompt to the effect of:

```
<shell> <shell program>
```

eg

```
bash helloworld
```

This simply instructs the shell to take a list of commands from a given file (your shell script). This method does not require the shell script to have execute permissions. However, in general you will execute your shell scripts via the first method.

And yet you may still find your script won''t execute - why? On some UNIX systems (Red Hat Linux included) the current directory (.) is not included in the **`PATH`** environment variable. This mans that the shell can''t find the script that you want to execute, even when it''s sitting in the current directory! To get around this either:

- Modify the PATH variable to include the "." directory:

```
PATH=$PATH:.
```

- Or, execute the program with an explicit path:

```
./helloworld
```

An Explanation of the Program

Line one, **`#!/bin/bash`** is used to indicate which shell the shell program is to be run in. If this program was written for the C shell, then you might have **`#!/bin/csh`** instead.

It is probably worth mentioning at this point that UNIX “executes” programs by first looking at the first two bytes of the file (this is similar to the way MS-DOS looks at the first two bytes of executable programs; all .EXE programs start with “MZ”). From these two characters, the system knows if the file is an interpreted script (**`#!`**) or some other file type (more information can be obtained about this by typing **`man file`**). If the file is an interpreted script, then the system looks for a following path indicating an interpreter. For example:

```
#!/bin/bash
#!/usr/bin/perl
#!/bin/sh
```

Are all valid interpreters.

Line two, **`# This is a program that says hello`**, is (you guessed it) a comment. The "#" in a shell script is interpreted as "anything to the right of this is a comment, go onto the next line". Note that it is similar to line one except that line one has the "`!`" mark after the comment.

Comments are a very important part of any program - it is a really good idea to include some. The reasons why are standard to all languages - readability, maintenance and self congratulation. It is more so important for a system administrator as they very rarely remain at one site for their entire working career, therefore, they must work with other people's shell scripts (as other people must work with theirs).

Always have a comment header; it should include things like:

```
# AUTHOR:       Who wrote it
# DATE:         Date first written
# PROGRAM:      Name of the program
# USAGE:        How to run the script; include any parameters
# PURPOSE:      Describe in more than three words what the
#               program does
#
# FILES:        Files the shell script uses
#
# NOTES:        Optional but can include a list of "features"
#               to be fixed
#
# HISTORY:      Revisions/Changes
```

This format isn't set in stone, but use common sense and write fairly self documenting programs.

Version Control Systems

Those of you studying software engineering may be familiar with the term, version control. Version control allows you to keep copies of files including a list of who made what changes and what those changes were. Version control systems can be very useful for keeping track of source code and is just about compulsory for any large programming project.

Linux comes with CVS (Concurrent Versions System) a widely used version control system. While version control may not seem all that important it can save a lot of heartache.

Many large sites will actually keep copies of system configuration files in a version control system.

Line three, **`echo "Hello $LOGNAME, I hope you have a nice day!"`** is actually a command. The **`echo`** command prints text to the screen. Normal shell rules for interpreting special characters apply for the **`echo`** statement, so you should generally enclose most text in "". The only tricky bit about this line is the **`$LOGNAME`** . What is this?

`$LOGNAME` is a shell variable; you can see it and others by typing "set" at the shell prompt. In the context of our program, the shell substitutes the **`$LOGNAME`** value with the username of the person running the program, so the output looks something like:

```
Hello jamiesob, I hope you have a nice day!
```

All variables are referenced for output by placing a "**`$`**" sign in front of them - we will examine this in the next section.

Exercises

8.1. Modify the **`helloworld`** program so its output is something similar to:
```
Hello <username>, welcome to <machine name>
```

All You Ever Wanted to Know About Variables

You have previously encountered shell variables and the way in which they are set. To quickly revise, variables may be set at the shell prompt by typing:

```
[david@faile david]$ variable="a string"
```

Since you can type this at the prompt, the same syntax applies within shell programs.

You can also set variables to the results of commands, for example:

```
[david@faile david]$ variable=‘ls -al‘
```

(Remember - the **‘** is the execute quote)

To print the contents of a variable, simply type:

```
[david@faile david]$ echo $variable
```

Note that we've added the "**$**" to the variable name. **Variables are always accessed for output with the "$" sign, but without it for input/set operations.**

Returning to the previous example, what would you expect to be the output?

You would probably expect the output from **ls -al** to be something like:

```
drwxr-xr-x   2 jamiesob users        1024 Feb 27 19:05 ./
drwxr-xr-x  45 jamiesob users        2048 Feb 25 20:32 ../
-rw-r--r--   1 jamiesob users         851 Feb 25 19:37 conX
-rw-r--r--   1 jamiesob users       12517 Feb 25 19:36 confile
-rw-r--r--   1 jamiesob users           8 Feb 26 22:50 helloworld
-rw-r--r--   1 jamiesob users       46604 Feb 25 19:34 net-acct
```

and therefore, printing a variable that contains the output from that command would contain something similar, yet you may be surprised to find that it looks something like:

```
drwxr-xr-x 2 jamiesob users 1024 Feb 27 19:05 ./ drwxr-xr-x 45
jamiesob users 2048 Feb 25 20:32 ../ -rw-r--r-- 1 jamiesob users 851
Feb 25 19:37 conX -rw-r--r-- 1 jamiesob users 12517 Feb 25 19:36
confile -rw-r--r-- 1 jamiesob users 8 Feb 26 22:50 helloworld -rw-r--
r-- 1 jamiesob users 46604 Feb 25 19:34 net-acct
```

Why?

When placing the output of a command into a shell variable, the shell removes all the end-of-line markers, leaving a string separated only by spaces. The use for this will become more obvious later, but for the moment, consider what the following script will do:

```
#!/bin/bash
$filelist=‘ls‘
cat $filelist
```

Exercise

> **8.2.** Type in the above program and run it. Explain what is happening. Would the above program work if "**ls -al**" was used rather than "**ls**" - Why/why not?

Predefined Variables

There are many predefined shell variables, most established during your login. Examples include **$LOGNAME**, **$HOSTNAME** and **$TERM** - these names are not always standard from system to system (for example, **$LOGNAME** can also be called **$USER**). There are however, several standard predefined shell variables you should be familiar with. These include:

```
$$    (The current process ID)
$?    (The exits status of last command)
```

How would these be useful?

$$

$$ is extremely useful in creating unique temporary files. You will often find the following in shell programs:

```
some command > /tmp/temp.$$
.
.
some commands using /tmp/temp.$$>
.
.
rm /tmp/temp.$$
```

/tmp/temp.$$ would always be a unique file - this allows several people to run the same shell script simultaneously. Since one of the only unique things about a process is its PID (Process-Identifier), this is an ideal component in a temporary file name. It should be noted at this point that temporary files are generally located in the **/tmp** directory.

$?

$? becomes important when you need to know if the last command that was executed was successful. All programs have a numeric exit status - on UNIX systems 0 indicates that the program was successful, any other number indicates a failure. We will examine how to use this value at a later point in time.

Is there a way you can show if your programs succeeded or failed? Yes! This is done via the use of the **exit** command. If placed as the last command in your shell program, it will enable you to indicate, to the calling program, the exit status of your script.

`exit` is used as follows:

```
exit 0            # Exit the script, $? = 0 (success)
exit 1            # Exit the script, $? = 1 (fail)
```

Another category of standard shell variables are shell parameters.

Parameters - Special Shell Variables

If you thought shell programming was the best thing since COBOL, then you haven't even begun to be awed - shell programs can actually take parameters. Table 8.1 lists each variable associated with parameters in shell programs:

Variable	Purpose
$0	the name of the shell program
$1 thru $9	the first thru to ninth parameters
$#	the number of parameters
$*	all the parameters passed represented as a single word with individual parameters separated
$@	all the parameters passed with each parameter as a separate word

Table 8.1
Shell Parameter Variables

The following program demonstrates a very basic use of parameters:

```
#!/bin/bash
# FILE:         parm1
VAL=‘expr ${1:-0} + ${2:-0} + ${3:-0}‘
echo "The answer is $VAL"
```

> ***Pop Quiz**: Why are we using **`${1:-0}`** instead of **`$1?`** Hint: What would happen if any of the variables were not set?*

A sample testing of the program looks like:

```
[david@faile david]$ parm1 2 3 5
The answer is  10

[david@faile david]$ parm1 2 3
The answer is  5

[david@faile david]$ parm
The answer is  0
```

Consider the program below:

```
#!/bin/bash
# FILE:         mywc

FCOUNT=‘ls $*  2> /dev/null | wc -w‘
echo "Performing word count on $*"
echo
wc -w $* 2> /dev/null
echo
echo "Attempted to count words on $# files, found $FCOUNT"
```

If the program that was run in a directory containing:

```
conX          net-acct      notes.txt     shellprog~    t1~
confile       netnasties    notes.txt~    study.htm     ttt
helloworld    netnasties~   scanit*       study.txt     tes/
my_file       netwatch      scanit~       study_~1.htm
mywc*         netwatch~     shellprog     parm1*
```

Some sample testing would produce:

```
[david@faile david]$ mywc mywc
Performing word count on mywc
```

```
34 mywc

Attempted to count words on 1 files, found       1
[david@faile david]$ mywc mywc anotherfile
Performing word count on mywc anotherfile

34 mywc
34 total


Attempted to count words on 2 files, found       1
```

Exercise

> 8.3. Explain line by line what this program is doing. What would happen if the user didn't enter any parameters? How could you fix this?

Only Nine Parameters?

Well that's what it looks like doesn't it? We have **$1** to **$9** - what happens if we try to access **$10?** Try the code below:

```
#!/bin/bash
# FILE:           testparms
echo "$1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12"
echo $*
echo $#
```

Run testparms as follows:

```
[david@faile david]$ testparms a b c d e f g h I j k l
```

The output will look something like:

```
a b c d e f g h i a0 a1 a2
a b c d e f g h I j k l
12
```

Why?

The shell only has 9 command-line parameters defined at any one time **$1** to **$9**. When the shell sees "**$10**" it interprets this as "**$1**" with a "**0**" after it. This is where $10 in the above results in a0. The a is the value of $1 with the 0 added.

On the otherhand **$*** allows you to see all the parameters you typed!

So how do you access $10, $11 etc. To our rescue comes the **shift** command. **shift** works by removing the first parameter from the parameter list and shuffling the parameters along. Thus **$2** becomes **$1**, **$3** becomes **$2** etc. Finally, (what was originally) the tenth parameter becomes **$9**. However, beware! Once you've run **shift**, you have lost the original value of **$1** forever - it is also removed from **$*** and $@. **shift** is executed by, well, placing the word "shift" in your shell script, for example:

```
#!/bin/bash
echo $1 $2 $3
```

```
shift
echo $1 $2 $3
```

Exercise

8.4. Modify the **testparms** program so the output looks something like:

a b c d e f g h i a0 a1 a2
a b c d e f g h I j k l
12
b c d e f g h i j b1 b2 b3
b c d e f g h i j k l
11
c d e f g h i j k c0 c1 c2
c d e f g h I j k l
10

The difference between *$** and *$@*

$* and $@ are very closely related. They both are expanded to become a list of all the command line parameters passed to a script. However, there are some subtle differences in how these two variables are treated. The subtleties are made even more difficult when they appear to act in a very similar way (in some situations). For example, let's see what happens with the following shell script

```
#for name in $*
for name in $@
do
  echo param is $name
done
```

The idea with this script is that you can test it with either $* or $@ by uncommenting the one you want to experiment and comment out the other line. The following examples show what happens when I run this script. The first time with $@, the second with $*

```
[david@faile david]$ tmp.sh hello "how are you" today 1 2 3
param is hello
param is how
param is are
param is you
param is today
param is 1
param is 2
param is 3
[david@faile david]$ tmp.sh hello "how are you" today 1 2 3
param is hello
param is how
param is are
param is you
param is today
param is 1
param is 2
param is 3
```

As you can see no difference!! So what's all this fuss with $@ and $*? The difference comes when $@ and $* are used within double quotes. In this situation they work as follows

- `$@`
  Is expanded to all the command-line parameters joined as a single word with usually a space seperating them (the separating character can be changed).
- `$*`
  Expands to all the command-line parameters BUT each command-line parameter is treated as if it is surrounded by double quotes "". This is especially important when one of the parameters contains a space.

Let's modify the our example script so that $@ and $* are surrounded by ""

```
#for name in "$*"
for name in "$@"
do
  echo param is $name
done
```

Now look at what happens when we run it using the same parameters as before. Again the $@ version is executed first then the $* version.

```
[david@faile david]$ tmp.sh hello "how are you" today 1 2 3
param is hello
param is how are you
param is today
param is 1
param is 2
param is 3
[david@faile david]$ tmp.sh hello "how are you" today 1 2 3
param is hello how are you today 1 2 3
```

With the second example, where $* is used, the difference is obvious. The first example, where $@ is used, shows the advantage of $@. The second parameter is maintained as a single parameter.

The basics of input/output (IO)

We have already encountered the "**`echo`**" command, yet this is only the "O" part of IO - how can we get user input into our programs? We use the "**`read`**" command. For example:

```
#!/bin/bash
# FILE:           testread
read X
echo "You said $X"
```

The purpose of this enormously exciting program should be obvious.

Just in case you were bored with the `echo` command. Table 8.2 shows a few backslash characters that you can use to brighten your shell scripts:

Character	Purpose
\a	alert (bell)
\b	backspace
\c	don't display the trailing newline
\n	new line
\r	carriage return
\t	horizontal tab
\v	vertical tab
\\	backslash
\nnn	the character with ASCII number nnn (octal)

Table 8.2
echo backslash options

(type "**man echo**" to see this exact table :)

To enable **echo** to interpret these backslash characters within a string, you must issue the **echo** command with a "**-e**" switch. You may also add a "**-n**" switch to stop **echo** printing a new-line at the end of the string - this is a good thing if you want to output a prompting string. For example:

```
#!/bin/bash
# FILE:           getname
echo -n "Please enter your name: "
read NAME
echo "Your name is $NAME"
```

(This program would be useful for those with a very short memory)

At the moment, we've only examined reading from STDIN (standard input a.k.a. the keyboard) and STDOUT (standard output a.k.a. the screen) - if we want to be really clever we can change this.

What do you think the following does?

```
read X < afile
```

or what about

```
echo $X > anotherfile
```

If you said that the first read the contents of `afile` into a variable **$X** and the second wrote the value of **$X** to `anotherfile` you'd almost be correct. The `read` operation will only read the first line (up to the end-of-line marker) from `afile` - it doesn't read the entire file.

You can also use the "**>>**" and "**<<**" redirection operators.

Exercises

8.5. What would you expect:

```
read X << END
```

would do? What do you think **$X** would hold if the input was:

```
Dear Sir
I have no idea why your computer blew up.
Kind regards, me.
END
```

And now for the hard bits

Scenario

So far we have been dealing with very simple examples - mainly due to the fact we've been dealing with very simple commands. Shell scripting was not invented so you could write programs that ask you your name then display it. For this reason, we are going to be developing a real program that has a useful purpose. We will do this section by section as we examine more shell programming concepts. While you are reading each section, you should consider how the information could assist in writing part of the program.

The actual problem is as follows:

You've been appointed as a system administrator to an academic department within a small (anonymous) regional university. The previous system administrator left in rather a hurry after it was found that department''s main server had being playing host to plethora of pornography, warez (pirate software) and documentation regarding interesting alternative uses for various farm chemicals.

There is some concern that the previous sys admin wasn't the only individual within the department who had been availing themselves to such wonderful and diverse resources on the Internet. You have been instructed to identify those persons who have been visiting "undesirable" Internet sites and advise them of the department's policy on accessing inappropriate material (apparently there isn't one, but you've been advised to improvise). Ideally, you will produce a report of people accessing restricted sites, exactly which sites and the number of times they visited them.

To assist you, a network monitoring program produces a datafile containing a list of users and sites they have accessed, an example of which is listed below:

```
FILE: netwatch

jamiesob    mucus.slime.com
tonsloye    xboys.funnet.com.fr
tonsloye    sweet.dreams.com
root  sniffer.gov.au
jamiesob    marvin.ls.tc.hk
jamiesob    never.land.nz
```

```
jamiesob    guppy.pond.cqu.edu.au
tonsloye    xboys.funnet.com.fr
tonsloye    www.sony.com
janesk      horseland.org.uk
root  www.nasa.gov
tonsloye    warez.under.gr
tonsloye    mucus.slime.com
root  ftp.ns.gov.au
tonsloye    xboys.funnet.com.fr
root  linx.fare.com
root  crackz.city.bmr.au
janesk      smurf.city.gov.au
jamiesob    mucus.slime.com
jamiesob    mucus.slime.com
```

After careful consideration (and many hours of painstaking research) a steering committee on the department's policy on accessing the internet has produced a list of sites that they have deemed "prohibited" - these sites are contained in a data file, an example of which is listed below:

```
FILE: netnasties


mucus.slime.com
xboys.funnet.com.fr
warez.under.gr
crackz.city.bmr.au
```

It is your task to develop a shell script that will fulfil these requirements (at the same time ignoring the privacy, ethics and censorship issues at hand :)

(Oh, it might also be an idea to get Yahoo! to remove the link to your main server under the /Computers/Software/Hackz/Warez/Sites listing... ;)

if ... then ... maybe?

Shell programming provides the ability to test the exit status from commands and act on them. One way this is facilitated is:

```
if command
then
  do other commands
fi
```

You may also provide an "alternate" action by using the "if" command in the following format:

```
if command
then
  do other commands
else
  do other commands
fi
```

And if you require even more complexity, you can issue the if command as:

```
if command
then
  do other commands
elif anothercommand
  do other commands
fi
```

To test these structures, you may wish to use the `true` and `false` UNIX commands. `true` always sets `$?` to 0 and `false` sets `$?` to 1 after executing.

Remember: **`if`** tests the exit code of a command - it isn't used to compare values; to do this, you must use the **`test`** command in combination with the **`if`** structure - **`test`** will be discussed in the next section.

What if you wanted to test the output of two commands? In this case, you can use the shell's && and `||` operators. These are effectively "smart" AND and OR operators.

The **`&&`** works as follows:

```
command1 && command2
```

`command2` will only be executed if **`command1`** succeeds.

The **`||`** works as follows:

```
command1 || command2
```

`command2` will only be executed if **`command1`** fails.

These are sometimes referred to as "short circuit" operators in other languages.

Given our problem, one of the first things we should do in our program is to check if our datafiles exist. How would we do this?

```
#!/bin/bash
# FILE:           scanit
if ls netwatch && ls netnasties
then
      echo "Found netwatch and netnasties!"
else
      echo "Can not find one of the data files - exiting"
      exit 1
fi
```

Exercise

> **8.6.** Enter the code above and run the program. Notice that the output from the **`ls`** commands (and the errors) appear on the screen - this isn't a very good thing. Modify the code so the only output to the screen is one of the **`echo`** messages.

Testing Testing...

Perhaps the most useful command available to shell programs is the **`test`** command. It is also the command that causes the most problems for first time shell programmers - the first program they ever write is usually (imaginatively) called `test` - they attempt to run it - and nothing happens - why? (Hint: type `which test`, then type `echo $PATH` - why does the system command `test` run before the programmer's shell script?)

The test command allows you to:

- test the length of a string

- compare two strings
- compare two numbers
- check on a file's type
- check on a file's permissions
- combine conditions together

`test` actually comes in two flavours:

```
test an_expression
```

and

```
[ an_expression ]
```

They are both the same thing - it's just that `[` is soft-linked to `/usr/bin/test` ; `test` actually checks to see what name it is being called by; if it is [then it expects a] at the end of the expression.

What do we mean by "expression"? The expression is the string you want evaluated. A simple example would be:

```
if [ "$1" = "hello" ]
then
   echo "hello to you too!"
else
   echo "hello anyway"
fi
```

This simply tests if the first parameter was `hello`. Note that the first line could have been written as:

```
if test "$1" = "hello"
```

Tip: Note that we surrounded the variable $1 in quotes. This is to take care of the case when $1 doesn't exist - in other words, there were no parameters passed. If we had simply put $1 and there wasn't any $1, then an error would have been displayed:

```
test: =: unary operator expected
```

This is because you'd be effectively executing:

```
test NOTHING = "hello"
```

`=` expects a string to its left and right - thus the error. However, when placed in double quotes, you be executing:

```
test "" = "hello"
```

which is fine; you're testing an empty string against another string.

You can also use `test` to tell if a variable has a value in it by:

```
test $var
```

This will return true if the variable has something in it, false if the variable doesn't exist OR it contains null ("").

We could use this in our program. If the user enters at least one username to check on, them we scan for that username, else we write an error to the screen and exit:

```
if [ $1 ]
then
  the_user_list=echo $*
else
  echo "No users entered - exiting!
  exit 2
fi
```

Expressions, expressions!

So far we've only examined expressions containing string based comparisons. The following tables list all the different types of comparisons you can perform with the `test` command.

Expression	True if
`-z string`	length of string is 0
`-n string`	length of string is not 0
`string1 = string2`	if the two strings are identical
`string != string2`	if the two strings are NOT identical
`String`	if string is not NULL

Table 8.3
String based tests

Expression	True if
`int1 -eq int2`	first int is equal to second
`int1 -ne int2`	first int is not equal to second
`int1 -gt int2`	first int is greater than second
`Int1 -ge int2`	first int is greater than or equal to second
`Int1 -lt int2`	first int is less than second
`Int1 -le int2`	first int is less than or equal to second

Table 8.4
Numeric tests

Expression	True if
`-r file`	`File` exists and is readable
`-w file`	`file` exists and is writable
`-x file`	`file` exists and is executable
`-f file`	`file` exists and is a regular file
`-d file`	`file` exists and is directory
`-h file`	`file` exists and is a symbolic link
`-c file`	`file` exists and is a character special file
`-b file`	`file` exists and is a block special file
`-p file`	`file` exists and is a named pipe
`-u file`	`file` exists and it is setuid
`-g file`	`file` exists and it is setgid
`-k file`	`file` exists and the sticky bit is set
`-s file`	`file` exists and its size is greater than 0

Table 8.5
File tests

Expression	Purpose
`!`	reverse the result of an expression
`-a`	AND operator
`-o`	OR operator
`( expr )`	group an expression, parentheses have special meaning to the shell so to use them in the test command you must quote them

Table 8.6
Logic operators with test

Remember: **`test`** uses different operators to compare strings and numbers - using **`-ne`** on a string comparison and **`!=`** on a numeric comparison is incorrect and will give undesirable results.

Exercise

8.7. Modify the code for **`scanit`** so it uses the **`test`** command to see if the datafiles exists.

All about `case`

Ok, so we know how to conditionally perform operations based on the return status of a command. However, like a combination between the `if` statement and the `test $string = $string2`, there exists the **`case`** statement.

```
case value in
  pattern 1)      command
            anothercommand ;;
  pattern 2)      command
                  anothercommand ;;
esac
```

case works by comparing value against the listed patterns. If a match is made, then the commands associated with that pattern are executed (up to the "**;;**" mark) and **$?** is set to 0. If a match isn't made by the end of the case statement (**esac**) then **$?** is set to 1.

The really useful thing is that wildcards can be used, as can the "|" symbol which acts as an OR operator. The following example gets a Yes/No response from a user, but will accept anything starting with "Y" or "y" as YES, "N" or "n" as no and anything else as "MAYBE"

```
echo -n "Your Answer: "
read ANSWER
case $ANSWER in
  Y* | y*) ANSWER="YES" ;;
  N* | n*) ANSWER="NO" ;;
  *) ANSWER="MAYBE" ;;
esac
echo $ANSWER
```

Exercise

8.8. Write a shell script that inputs a date and converts it into a long date form. For example:
$~ > mydate 12/3/97
12th of March 1997

$~ > mydate
Enter the date: 1/11/74
1st of November 1974

Loops and Repeated Action Commands

Looping - "*the exciting process of doing something more than once*" - and shell programming allows it. There are three constructs that implement looping:

```
while - do - done
for - do - done
until - do - done
```

while

The format of the **while** construct is:

```
while command
do
  commands
done
```

(while command is true, commands are executed)

Example

```
while [ $1 ]
do
  echo $1
  shift
done
```

What does this segment of code do? Try running a script containing this code with **`a b c d e`** on the command line.

`while` also allows the redirection of input. Consider the following:

```
#!/bin/bash
# FILE:           linelist
#
count=0
while read BUFFER
do
  count=‘expr $count + 1‘     # Increment the count
  echo "$count $BUFFER"       # Echo it out
done < $1               # Take input from the file
```

This program reads a file line by line and echo’s it to the screen with a line number.

Given our **`scanit`** program, the following could be used read the netwatch datafile and compare the username with the entries in the datafile:

```
while read buffer
do
  user=‘echo $buffer | cut -d" " -f1‘
  site=‘echo $buffer | cut -d" " -f2‘
  if [ "$user" = "$1" ]
  then
    echo "$user visited $site"
  fi
done < netwatch
```

Exercise

> **8.9.** Modify the above code so that the site is compared with all sites in the prohibited sites file (**`netnasties`**). Do this by using another **`while`** loop. If the user has visited a prohibited site, then **`echo`** a message to the screen.

`for`

The format of the **`for`** construct is:

```
for variable in list_of_variables
do
  commands
done
```

(for each value in list_of_variables, "commands" are executed)

Example

We saw earlier in this chapter examples of the for command showing the difference between $* and $@.

Another example

```
for count in 10 9 8 7 6 5 4 3 2 1
do
  echo -n "$count.."
done

echo
```

Modifying *scanit*

Given our **scanit** program, we might wish to report on a number of users. The following modifications will allow us to accept and process multiple users from the command line:

```
for checkuser in $*
do
  while read buffer
  do
    while read checksite
    do
      user=‘echo $buffer | cut -d" " -f1‘
      site=‘echo $buffer | cut -d" " -f2‘
      if [ "$user" = "$checkuser" -a "$site" = "$checksite" ]
      then
        echo "$user visited the prohibited site $site"
      fi
    done < netnasties
  done < netwatch
done
```

Problems with running scanit

A student in the 1999 offering of 85321 reported the following problem with the scanit program on page 160 of chapter 8 of the 85321 textbook.

When running her program she types

```
Bash scanit jamiesob
```

and quite contrary to expectations she gets 80 lines of output that includes

```
root visited the prohibited site crackz.city.bmr.au
root visited the prohibited site crackz.city.bmr.au
janesk visited the prohibited site smurf.city.gov.au
janesk visited the prohibited site smurf.city.gov.au
janesk visited the prohibited site smurf.city.gov.au
janesk visited the prohibited site smurf.city.gov.au
jamiesob visited the prohibited site mucus.slime.com
jamiesob visited the prohibited site mucus.slime.com
```

If everything is working the output you should get is three lines of code reporting that the user jamiesob has visited the site mucus.slime.com.

So what is the problem?

Well let's have a look at her shell program

```
for checkuser in $*
do
  while read buffer
  do
    while read checksite
    do
      user=‘echo $buffer | cut -d" " -f1‘
      site=‘echo $buffer | cut -d" " -f2‘
      if [ "$user"="$checkuser" -a "$site"="$checksite" ]
      then
        echo "$user visited the prohibited site $site"
      fi
    done < netnasties
  done < netwatch
done
```

Can you see the problem?

How do we identify the problem? Well let's start by thinking about what the problem is. The problem is that it is showing too many lines. The script is not excluding lines which should not be displayed. Where are the lines displayed?

The only place is within the if command. This seems to imply that the problem is that the if command isn't working. It is matching too many times, in fact it is matching all of the lines.

The problem is that if command is wrong or not working as expected.

How is it wrong?

Common mistakes with the if command include

- not using the test command
  Some people try comparing "things" without using the test command
  `if "$user"="$checkuser" -a "$site"="$checksite"`
  The student is using the test command in our example. In fact, she is using the [form of the test command. So this isn't the problem.

- using the wrong comparison operator
  Some people try things like
  `if [ "$user" == "$checkuser" ]` or
  `if [ "$user" -eq "$checkuser" ]`
  Trouble with this is that == is comparison operator from the C/C++ programming languages and not a comparison operator supported by the test command. `-eq` is a comparison operator supported by test but it is used to compare numbers not strings. This isn't the problem here.

The problem here is some missing spaces around the = signs.

Remember that `[` is actually a shell command (it's the same command `test`). Like other commands it takes parameters. Let's have a look at the parameters that the test command takes in this example program.

The `test` command is
`[ "$user"="$checkuser" -a "$site"="$checksite" ]`

Parameters must be surrounded by spaces. So this command has four parameters (the first [is the command name)

1. "$user"="$checkuser"
2. -a
3. "$site"="$checksite"
4.]

By now you might start to see the problem. For the test command to actual compare two "things" it needs to see the = as a separate parameter. The problem is that because there are no spaces around the = characters in this test command the = is never seen. It's just part of a string.

The solution to this problem is to put some space characters around the two =. So we get

```
[ "$user" = "$checkuser" -a "$site" = "$checksite" ]
```

So what is happening

So what is actually happening? Why is the test always returning true. We know this because the script displays a line for all the users and all the sites.

To find the solution to this problem we need to take a look at the manual page for the test command. On current Linux computers you can type man test and you will see a manual page for this command. However, it isn't the one you should look at.

Type the following command which test. It should tell you where the executable program for test is located. Trouble is that on current Linux computers it won't. That's because there isn't one. Instead the test command is actually provided by the shell, in this case bash. To find out about the test command you need to look at the man page for bash.

The other approach would be to look at Table 8.3 from chapter 8 of the 85321 textbook. In particular the last entry which says that if the expression in a test command is a string then the test command will return true if the string is non-zero (i.e. it has some characters).

Here are some examples to show what this actually means.

In these examples I'm using the test command by itself and then using the echo command to have a look at the value of the $? shell variable. The $? shell variable holds the return status of the previous command.

For the test command if the return status is 0 then the expression was true. If it is 1 then the expression as false.

```
[david@faile 8]$ [ fred ]
[david@faile 8]$ echo $?
0
[david@faile 8]$ [ ]
[david@faile 8]$ echo $?
1
[david@faile 8]$ [ "jamiesob"="mucus.slime.com" ]
[david@faile 8]$ echo $?
0
```

```
[david@faile 8]$ [ "jamiesob" = "mucus.slime.com" ]
[david@faile 8]$ echo $?
1
```

In the first example the expression is fred a string with a non-zero length. So the return status is 0 indicating true. In the second example there is no expression, so it is a string with zero length. So the return status is 1 indicating false.

The last two examples are similar to the problem and solution in the student's program. The third example is similar to the students problem. The parameter is a single non-zero length string ("jamiesob"="mucus.slime.com") so the return status is 0 indicating truth.

When we add the spaces around the = we finally get what we wanted. The test command actually compares the two strings and sets the return status accordingly and because the strings are different the return status is 1 indicating false.

So what about the -a operator used in the student's program. Well the -a simply takes the results of two expressions (one on either side) and ands them together. In the student's script there the two expressions are non-zero length strings. Which are always true. So that becomes 0 -a 0 (TRUE and TRUE) which is always true.

Here are some more examples

```
[david@faile 8]$ [ "jamiesob"="mucus.slime.com" -a "david"="fred" ]
[david@faile 8]$ echo $?
0
[david@faile 8]$ [ "jamiesob"="mucus.slime.com" -a "" ]
[david@faile 8]$ echo $?
1
[david@faile 8]$ [ "jamiesob" = "mucus.slime.com" -a "david" =
"david" ]
[david@faile 8]$ echo $?
1
[david@faile 8]$ [ "jamiesob" = "jamiesob" -a "david" = "david" ]
[david@faile 8]$ echo $?
0
```

The first example here is what is happening in the student's program. Two non-zero length strings, which are always true, "anded" together will always return true regardless of the strings.

The second example shows what happens when one side of the -a is a zero length string. A zero length string is always false, false and true is always false, so this example has a return status of 1 indicating false.

The last two examples show "working" versions of the test command with spaces in all the right places. Where the two strings being compared are different the comparison is false and the test command is returning false. Where the two strings being compared are the same the comparison operator is returning true and the test command is returning true.

Exercises

8.10. What will be the return status of the following `test` commands? Why?

```
["hello"]
[ $HOME ]
[ "‘hello‘" ]
```

8.11. The above code is very inefficient IO wise - for every entry in the **`netwatch`** file, the entire **`netnasties`** file is read in. Modify the code so that the while loop reading the netnasties file is replaced by a for loop. (Hint: what does: `BADSITES=‘cat netnasties‘` do?)

EXTENSION: What other IO inefficiencies does the code have? Fix them.

Speed and shell scripts

Exercise 8.11 is actually a very important problem in that it highlights a common mistake made by many novice shell programmers. This mistake is especially prevalent amongst people who have experience in an existing programming language like C/C++ or Pascal.

This supplementary material is intended to address that problem and hopefully make it a little easier for you to answer question 11. Online lecture 8, particularly on slide 21 also addresses this problem. You might want to have a look at and listen to this slide before going much further.

What’s the mistake

A common mistake for beginning shell programmers make is to write shell programs as if they were C/C++ programs. In particular they tend not to make use of the collection of very good commands which are available.

Let’s take a look at a simple example of what I mean. The problem is to count the number of lines in a file (the file is called the_file). The following section discusses three solutions to this problem: a solution in C, a shell solution written like the C program, and a "proper" shell/UNIX solution

Solution in C

```
#include <stdio.h>

void main( void )
{
  int line_count = 0;
  FILE *infile;
  char line[500];

  infile = fopen( "the_file", "r" );

  while ( ! feof( infile ) )
  {
    fgets( line, 500, infile );
    line_count++;
  }
```

```
  printf( "Number of lines is %d\n", line_count-1 );
}
```

Pretty simple to understand? Open the file, read the file line by line, increment a variable for each line and then display the variable when we reach the end of the file.

Shell solution written by C programmer

It is common for new comers to the shell to write shell scripts like C (or whatever procedural language they are familiar with) programs. Here's a shell version of the previous C solution. It uses the same algorithm.

```
count=0
while read line
do
  count=‘expr $count + 1‘
done < the_file

echo Number of lines is $count
```

This shell script reads the file line by line, increment a variable for each line and when we reach the end of the file display the value.

Shell solution by shell programmer

Anyone with a modicum of UNIX experience will know that you don't need to write a shell program to solve this problem. You just use the wc command.

```
wc -l the_file
```

This may appear to be a fairly trivial example. However, it does emphasise a very important point. You don't want to use the shell commands like a normal procedural programming language. You want to make use of the available UNIX commands where ever possible.

Comparing the solutions

Let's compare the solutions.

The C program is obviously the longest solution when it comes to size of the program. The shell script is much shorter. The shell takes care of a lot of tasks you have to do with C and the use of wc is by far the shortest. The UNIX solutions are also much faster to write as there is no need for a compile/test cycle. This is one of the advantages of scripting languages like the shell, Perl and TCL.

What about speed of execution?

As we've seen in earlier chapters you can test the speed of executable programs (in a very coarse way) with the time command. The following shows the time taken for each solution. In the tests each of the three solutions worked on the same file which contained 1911 lines.

```
[david@faile david]$ time ./cprogram
Number of lines is 1911
0.00user 0.01system 0:00.01elapsed 83%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (79major+11minor)pagefaults 0swaps
```

```
[david@faile david]$ time sh shsolution
Number of lines is 1911
12.24user 14.17system 0:28.12elapsed 93%CPU (0avgtext+0avgdata
0maxresident)k
0inputs+0outputs (164520major+109070minor)pagefaults 0swaps

[david@faile david]$ time wc -l /var/log/messages
   1911 /var/log/messages
0.00user 0.01system 0:00.04elapsed 23%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (85major+14minor)pagefaults 0swaps
```

The lesson to draw from these figures is that solutions using the C program and the wc command have the same efficiency but using the wc command is much quicker.

The shell programming solution which was written like a C program is horrendously inefficient. It is tens of thousands of times slower than the other two solutions and uses an enormous amount of resources.

The problem

Obviously using while loops to read a file line by line in a shell program is inefficient and should be avoided. However, if you think like a C programmer you don't know any different.

When writing shell programs you need to modify how you program to make use of the strengths and avoid the weaknesses of shell scripting. Where possible you should use existing UNIX commands.

A solution for scanit?

Just because the current implementation of scanit uses two while loops it doesn't mean that your solution has to. Think about the problem you have to solve.

In the case of improving the efficiency of scanit you have to do the following

- for every user entered as a command line parameter
- see if the user has visited one of the sites listed in the netnasties file

To word it another way, you are searching for lines in a file which match a certain criteria. What UNIX command does that?

Number of processes

Another factor to keep in mind is the number of processes your shell script creates. Every UNIX command in a shell script will create a new process. Creating a new process is quite a time and resource consuming job performed by the operating system. One thing you want to do is to reduce the number of new processes created.

Let's take a look at the shell program solution to our problem

```
count=0
while read line
do
  count=‘expr $count + 1‘
done < the_file
```

```
echo Number of lines is $count
```

For a file with 1911 lines this shell program is going to create about 1913 processes. 1 process for the echo command at the end, one process to for a new shell to run the script and 1911 processes for the expr command. Every time the script reads a line it will create a new process to run the expr command. So the longer the file the less efficient this script is going to get.

One way to address this problem somewhat is to use the support that the bash shell provides for arithmetic. By using the shell's arithmetic functions we can avoid creating a new process because the shell process will do it.

Our new shell script looks like this

```
count=0
while read line
do
  count=$[ $count + 1 ]
done < /var/log/messages

echo Number of lines is $count
```

See the change in the line incrementing the count variable. It's now using the shell arithmetic support. Look what happens to the speed of execution.

```
[david@faile 8]$ time bash test6
Number of lines is 1915
1.28user 0.52system 0:01.83elapsed 98%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (179major+30minor)pagefaults 0swaps
```

We have a slightly bigger file but even so the speed is much, much better. However, the speed is still no where as good as simply using the wc command.

until

The format of the **`until`** construct is:

```
until command
do
  commands
done
```

("commands" are executed until "command" is true)

Example

```
until [ "$1" = "" ]
do
  echo $1
  shift
done
```

break and **continue**

Occasionally you will want to jump out of a loop; to do this you need to use the **break** command. **break** is executed in the form:

```
break
```

or

```
break n
```

The first form simply stops the loop, for example:

```
while true
do
  read BUFFER
  if [ "$BUFFER" = "" ]
  then
    break
  fi
  echo $BUFFER
done
```

This code takes a line from the user and prints it until the user enters a blank line. The second form of **break**, **break n** (where **n** is a number) effectively works by executing **break** "**n**" times. This can break you out of embedded loops, for example:

```
for file in $*
do
  while read BUFFER
  do
    if [ "$BUFFER" = "ABORT" ]
    then
      break 2
    fi
  echo $BUFFER
  done < $file
done
```

This code prints the contents of multiple files, but if it encounters a line containing the word "**ABORT**" in any one of the files, it stops processing.

Like **break**, **continue** is used to alter the looping process. However, unlike **break**, **continue** keeps the looping process going; it just fails to finish the remainder of the current loop by returning to the top of the loop. For example:

```
while read BUFFER
do
  charcount=‘echo $BUFFER | wc -c | cut -f1‘
  if [ $charcount -gt 80 ]
  then
    continue
  fi
  echo $BUFFER
done < $1
```

This code segment reads and echo’s the contents of a file - however, it does not print lines that are over 80 characters long.

Redirection

Not just the `while - do - done` loops can have IO redirection; it is possible to perform piping, output to files and input from files on `if`, `for` and `until` as well. For example:

```
if true
then
  read x
  read y
  read x
fi < afile
```

This code will read the first three lines from `afile`. Pipes can also be used:

```
read BUFFER
while [ "$BUFFER" != "" ]
do
  echo $BUFFER
  read BUFFER
done | todos > tmp.$$
```

This code uses a non-standard command called `todos`. `todos` converts UNIX text files to DOS textfiles by making the EOL (End-Of-Line) character equivalent to CR (Carriage-Return) LF (Line-Feed). This code takes STDIN (until the user enters a blank line) and pipes it into todos, which in turn converts it to a DOS style text file (**`tmp.$$`**) . In all, a totally useless program, but it does demonstrate the possibilities of piping.

Now for the really hard bits

Functional Functions

A symptom of most usable programming languages is the existence of functions. Theoretically, functions provide the ability to break your code into reusable, logical compartments that are the by product of top-down design. In practice, they vastly improve the readability of shell programs, making it easier to modify and debug them.

An alternative to functions is the grouping of code into separate shell scripts and calling these from your program. This isn't as efficient as functions, as functions are executed in the same process that they were called from; however other shell programs are launched in a separate process space - this is inefficient on memory and CPU resources.

You may have noticed that our **`scanit`** program has grown to around 30 lines of code. While this is quite manageable, we will make some major changes later that really require the "modular" approach of functions.

Shell functions are declared as:

```
function_name()
{
  somecommands
}
```

Functions are called by:

```
function_name parameter_list
```

YES! Shell functions support parameters. **$1** to **$9** represent the first nine parameters passed to the function and **$*** represents the entire parameter list. The value of **$0** isn't changed. For example:

```
#!/bin/bash
# FILE:         catfiles

catfile()
{
  for file in $*
  do
    cat $file
  done
}

FILELIST=‘ls $1‘
cd $1

catfile $FILELIST
```

This is a highly useless example (**cat *** would do the same thing) but you can see how the "main" program calls the function.

local

Shell functions also support the concept of declaring "local" variables. The `local` command is used to do this. For example:

```
#!/bin/bash

testvars()
{
  local localX="testvars localX"
  X="testvars X"
  local GlobalX="testvars GlobalX"
  echo "testvars: localX= $localX X= $X GlobalX= $GlobalX"
}

X="Main X"
GlobalX="Main GLobalX"
echo "Main 1: localX= $localX X= $X GlobalX= $GlobalX"

testvars

echo "Main 2: localX= $localX X= $X GlobalX= $GlobalX"
```

The output looks like:

```
Main 1: localX=  X= Main X GlobalX= Main GLobalX
testvars: localX= testvars localX X= testvars X GlobalX= testvars GlobalX
Main 2: localX=  X= testvars X GlobalX= Main GLobalX
```

The `return` trip

After calling a shell function, the value of **`$?`** is set to the exit status of the last command executed in the shell script. If you want to explicitly set this, you can use the **`return`** command:

```
return n
```

(Where **`n`** is a number)

This allows for code like:

```
if function1
then
  do_this
else
  do_that
fi
```

For example, we can introduce our first function into our **`scanit`** program by placing our datafile tests into a function:

```
#!/bin/bash
# FILE:           scanit
#

check_data_files()
{
  if [ -r netwatch -a -r netnasties ]
  then
    return 0
  else
    return 1
  fi
}

# Main Program

if check_data_files
then
  echo "Datafiles found"
else
  echo "One of the datafiles missing - exiting"
  exit 1
fi

# our other work...
```

Difficult and not compulsory

The following section (up to the section titled "Bugs and Debugging") is not compulsory for students studying 85321.

Recursion: (see "Recursion")

Shell programming even supports recursion. Typically, recursion is used to process tree-like data structures - the following example illustrates this:

```
#!/bin/bash
# FILE:           wctree

wcfiles()
{
  local BASEDIR=$1            # Set the local base directory
  local LOCALDIR=‘pwd‘  # Where are we?
  cd $BASEDIR                 # Go to this directory (down)
  local filelist=‘ls‘         # Get the files in this directory
  for file in $filelist
  do
    if [ -d $file ]           # If we are a directory, recurs
    then
      # we are a directory
      wcfiles "$BASEDIR/$file"
    else
      fc=‘wc -w < $file‘      # do word count and echo info
      echo "$BASEDIR/$file $fc words"
    fi
  done
  cd $LOCALDIR          # Go back up to the calling directory
}

if [ $1 ]               # Default to . if no parms
then
  wcfile $1
else
  wcfile "."
fi
```

Exercise

> **8.12.** What does the `wctree` program do? Why are certain variables declared as `local`? What would happen if they were not? Modify the program so it will only "recurs" 3 times.
>
> **EXTENSION:** There is actually a UNIX command that will do the same thing as this shell script - what is it? What would be the command line? (Hint: **`man find`**)

`wait`’ing and *`trap`*’ing

So far we have only examined linear, single process shell script examples. What if you want to have more than one action occurring at once? As you are aware, it is possible to launch programs to run in the background by placing an ampersand behind the command, for example:

```
runcommand &
```

You can also do this in your shell programs. It is occasionally useful to send a time consuming task to the background and proceed with your processing. An example of this would be a sort on a large file:

```
sort $largefile > $newfile &
do_a_function
do_another_funtion $newfile
```

The problem is, what if the sort hadn't finished by the time you wanted to use $newfile? The shell handles this by providing **`wait`** :

```
sort $largefile > $newfile &
do_a_function
wait
do_another_funtion $newfile
```

When **`wait`** is encountered, processing stops and "waits" until the child process returns, then proceeds on with the program. But what if you had launched several processes in the background? The shell provides the shell variable **`$!`** (the PID of the child process launched) which can be given as a parameter to wait - effectively saying "wait for this PID". For example:

```
sort $largefile1 > $newfile1 &
$SortPID1=$!
sort $largefile2 > $newfile2 &
$SortPID2=$!
sort $largefile3 > $newfile3 &
$SortPID3=$!
do_a_function
wait $SortPID1
do_another_funtion $newfile1
wait $SortPID2
do_another_funtion $newfile2
wait $SortPID3
do_another_funtion $newfile3
```

Another useful command is `trap`. `trap` works by associating a set of commands with a signal from the operating system. You will probably be familiar with:

```
kill -9 PID
```

which is used to kill a process. This command is in fact sending the signal "9" to the process given by PID. Available signals are shown in Table 8.7.

While you can't actually trap signal 9, you can trap the others. This is useful in shell programs when you want to make sure your program exits gracefully in the event of a shutdown (or some such event) (often you will want to remove temporary files the program has created). The syntax of using `trap` is:

```
trap commands signals
```

For example:

```
trap "rm /tmp/temp.$$" 1 2
```

will trap signals 1 and 2 - whenever these signals occur, processing will be suspended and the **`rm`** command will be executed.

You can also list every trap'ed signal by issuing the command:

```
trap
```

To "un-trap" a signal, you must issue the command:

```
trap "" signals
```

Signal	Meaning
0	Exit from the shell
1	Hangup
2	Interrupt
3	Quit
4	Illegal Instruction
5	Trace trap
6	IOT instruction
7	EMT instruction
8	Floating point exception
10	Bus error
12	Bad argument
13	Pipe write error
14	Alarm
15	Software termination signal

Table 8.7
UNIX signals

(Taken from "*UNIX Shell Programming*" Kochan et al)

The following is a somewhat clumsy form of IPC (Inter-Process Communication) that relies on **`trap`** and **`wait`**:

```
#!/bin/bash
# FILE:     saymsg
# USAGE: saymsg <create number of children> [total number of
#        children]

readmsg()
{
  read line < $$ # read a line from the file given by the PID
  echo "$ID - got $line!"     # of my *this* process ($$)
  if [ $CHILD ]
  then
    writemsg $line      # if I have children, send them message
  fi
}

writemsg()
{
  echo $* > $CHILD      # Write line to the file given by PID
  kill -1 $CHILD        # of my child.  Then signal the child.
}

stop()
{
  kill -15 $CHILD       # tell my child to stop
  if [ $CHILD ]
```

```
  then
    wait $CHILD         # wait until they are dead
    rm $CHILD                 # remove the message file
  fi
  exit 0
}


# Main Program

if [ $# -eq 1 ]
then
  NUMCHILD=‘expr $1 - 1‘
  saymsg $NUMCHILD $1 & # Launch another child
  CHILD=$!
  ID=0
  touch $CHILD                # Create empty message file
  echo "I am the parent and have child $CHILD"
else
  if [ $1 -ne 0 ]       # Must I create children?
  then
    NUMCHILD=‘expr $1 - 1‘    # Yep, deduct one from the number
    saymsg $NUMCHILD $2 &     # to be created, then launch them
    CHILD=$!
    ID=‘expr $2 - $1‘
    touch $CHILD        # Create empty message file
    echo "I am $ID and have child $CHILD"
  else
    ID=‘expr $2 - $1‘         # I don’t need to create children
    echo "I am $ID and am the last child"
  fi
fi

trap "readmsg" 1        # Trap the read signal
trap "stop" 15                # Trap the drop-dead signal

if [ $# -eq 1 ]         # If I have one parameter,
then                    # then I am the parent - I just read
  read BUFFER                 # STDIN and pass the message on
  while [ "$BUFFER" ]
  do
    writemsg $BUFFER
    read BUFFER
  done
  echo "Parent - Stopping"
  stop
else                    # Else I am the child who does nothing -
  while true                  # I am totally driven by signals.
  do
    true
  done
fi
```

So what is happening here? It may help if you look at the output:

```
psyche:~/sanotes[david@faile david]$ saymsg 3
I am the parent and have child 8090
I am 1 and have child 8094
I am 2 and have child 8109
I am 3 and am the last child
this is the first thing I type
1 - got this is the first thing I type!
```

```
2 - got this is the first thing I type!
3 - got this is the first thing I type!

Parent - Stopping

psyche:~/sanotes[david@faile david]$
```

Initially, the parent program starts, accepting a number of children to create. The parent then launches another program, passing it the remaining number of children to create and the total number of children. This happens on every launch of the program until there are no more children to launch.

From this point onwards the program works rather like Chinese whispers - the parent accepts a string from the user which it then passes to its child by sending a signal to the child - the signal is caught by the child and **`readmsg`** is executed. The child writes the message to the screen, then passes the message to its child (if it has one) by signalling it and so on and so on. The messages are passed by being written to files - the parent writes the message into a file named by the PID of the child process.

When the user enters a blank line, the parent process sends a signal to its child - the signal is caught by the child and **`stop`** is executed. The child then sends a message to its child to stop, and so on and so on down the line. The parent process can't exit until all the children have exited.

This is a very contrived example - but it does show how processes (even at a shell programming level) can communicate. It also demonstrates how you can give a function name to **`trap`** (instead of a command set).

Exercise

8.13. **`saymsg`** is riddled with problems - there isn't any checking on the parent process command line parameters (what if there wasn't any?) and it isn't very well commented or written - make modifications to fix these problems. What other problems can you see?

EXTENSION: Fundamentally **`saymsg`** isn't implementing very safe inter-process communication - how could this be fixed? Remember, one of the *main* problems concerning IPC is the race condition - could this happen?

Bugs and Debugging

If by now you have typed every example program in, completed every exercise and have not encountered one single error then you are a truly amazing person. However, if you are like me, you would have made at least 70 billion mistakes/ typos or TSE's (totally stupid errors) - and now I tell you the easy way to find them!

Method 1 - `set`

Issuing the truly inspired command of:

```
set -x
```

within your program will do wonderful things. As your program executes, each code line will be printed to the screen - that way you can find your mistakes, err, well, a little bit quicker. Turning tracing off is a good idea once your program works - this is done by:

```
set +x
```

Method 2 - `echo`

Placing a few **`echo`** statements in your code during your debugging is one of the easiest ways to find errors - for the most part this will be the quickest way of detecting if variables are being set correctly.

Very Common Mistakes

```
$VAR=‘ls‘
```

This should be `VAR=‘ls‘`. When setting the value of a shell variable you don't use the $ sign.

```
read $BUFFER
```

The same thing here. When setting the value of a variable you don't use the $ sign.

```
VAR=‘ls -al"
```

The second ‘ is missing

```
if [ $VAR ]
then
    echo $VAR
fi
```

Haven't specified what is being tested here. Need to refer to the contents of Tables 8.2 through 8.5

```
if [ $VAR -eq $VAR2 ]
then
   echo $VAR
fi
```

If $VAR and $VAR2 are strings then you can't use `–eq` to compare their values. You need to use `=`.

```
if [ $VAR = $VAR2 ] then
  echo $VAR
fi
```

The `then` must be on a separate line.

And now for the really really hard bits

Writing good shell programs

We have covered most of the theory involved with shell programming, but there is more to shell programming than syntax. In this section, we will complete the **scanit** program, examining efficiency and structure considerations.

scanit currently consists of one chunk of code with one small function. In its current form, it doesn't meet the requirements specified:

"...you will produce a report of people accessing restricted sites, exactly which sites and the number of times they visited them."

Our code, as it is, looks like:

```
#!/bin/bash
# FILE:           scanit
#

check_data_files()
{
  if [ -r netwatch -a -r netnasties ]
  then
    return 0
  else
    return 1
  fi
}

# Main Program

if check_data_files
then
  echo "Datafiles found"
else
  echo "One of the datafiles missing - exiting"
  exit 1
fi

for checkuser in $*
do
  while read buffer
  do
    while read checksite
    do
      user=‘echo $buffer | cut -d" " -f1‘
      site=‘echo $buffer | cut -d" " -f2‘
      if [ "$user" = "$checkuser" -a "$site" = "$checksite" ]
      then
        echo "$user visited the prohibited site $site"
      fi
    done < netnasties
  done < netwatch
done
```

At the moment, we simply print out the user and site combination - no count provided. To be really effective, we should parse the file containing the

user/site combinations (**`netwatch`**), register and user/prohibited site combinations and then when we have all the combinations and count per combination, produce a report. Given our datafile checking function, the pseudo code might look like:

```
if data_files_exist
  ...
else
  exit 1
fi
check_netwatch_file
produce_report
exit
```

It might also be an idea to build in a "default" - if no username(s) are given on the command line, we go and get all the users from the **`/etc/passwd`** file:

```
f [ $1 ]
then
  the_user_list=$*
else
  get_passwd_users
fi
```

Exercise

> **8.14.** Write the shell function **`get_passwd_users`**. This function goes through the **`/etc/passwd`** file and creates a list of usernames. (Hint: username is field one of the password file, the delimiter is "**`:`**")

`eval` the wonderful!

The use of `eval` is perhaps one of the more difficult concepts in shell programming to grasp is the use of **`eval`**. **`eval`** effectively says “parse (or execute) the following twice”. What this means is that any shell variables that appear in the string are “substituted” with their real value on the first parse, then used as-they-are for the second parse.

The use of this is difficult to explain without an example, so we’ll refer back to our case study problem.

The real challenge to this program is how to actually store a count of the user and site combination. The following is how I'd do it:

```
checkfile()
{
  # Goes through the netwatch file and saves user/site
  # combinations involving sites that are in the "restricted"
  # list

  while read buffer
  do
    username=‘echo $buffer | cut -d" " -f1‘     # Get the username
    # Remove “.”’s from the string
    site=`echo $buffer | cut -d" " -f2 | sed s/\\\.//g`
    for checksite in $badsites
    do
      checksite=`echo $checksite | sed s/\\\.//g`
```

```
      # Do this for the compare sites
      if [ "$site" = "$checksite" ]
      then
        usersite="$username$checksite"
        # Does the VARIABLE called $usersite exist? Note use of eval
        if eval [ \$$usersite ]
        then
          eval $usersite=\‘expr \$$usersite + 1\‘
        else
          eval $usersite=1
        fi
      fi
    done
  done < netwatch
}
```

There are only two *really* tricky lines in this function:

```
1.    site=‘echo $buffer | cut -d" " -f2 | sed s/\\\./g‘
```

Creates a variable `site`; if buffer (one line of netwatch) contained

```
rabid.dog.com
```

then `site` would become:

```
rabiddogcom
```

The reason for this is because of the variable `usersite`:

```
usersite="$username$checksite"
```

What we are actually creating is a variable name, stored in the variable usersite - why (you still ask) did we remove the "."'s? This becomes clearer when we examine the second tricky line:

```
2.    eval $usersite=\‘expr \$$usersite + 1\‘
```

Remember eval "double" or "pre" parses a line - after eval has been run, you get a line which looks something like:

```
# $user="jamiesobrabiddogcom"
jamiesobrabiddogcom=‘expr $jamiesobrabiddogcom + 1‘
```

What should become clearer is this: the function reads each line of the `netwatch` file. If the site in the `netwatch` file matches one of the sites stored in `netnasties` file (which has been cat'ed into the variable `badsites`) then we store the user/site combination. We do this by first checking if there exists a variable by the name of the user/site combination - if one does exist, we add 1 to the value stored in the variable. If there wasn't a variable with the name of the user/site combination, then we create one by assigning it to "1".

At the end of the function, we should have variables in memory for all the user/prohibited site combinations found in the netwatch file, something like:

```
jamiesobmucusslimecom=3
tonsloyemucusslimecom=1
tonsloyeboysfunnetcomfr=3
tonsloyewarezundergr=1
rootwarzundergr=4
```

Note that this would be the case even if we were only interested in the users `root` and `jamiesob`. So why didn't we check in the function if the user in the netwatch file was one of the users we were interested in? Why should we!? All that does is adds an extra loop:

```
for every line in the file
  for every site
     for every user
     do check
     create variable if user and if site in userlist,
     badsitelist
```

whereas all we have now is

```
for every line in the file
    for every site
       create variable if site in badsitelist
```

We are still going to have to go through every user/badsite combination anyway when we produce the report - why add the extra complexity?

You might also note that there is minimal file IO - datafiles are only read ONCE - lists (memory structures) may be read more than once.

Exercise

8.15. Given the `checksite` function, write a function called `produce_report` that accepts a list of usernames and finds all user/badsite combinations stored by `checkfile`. This function should `echo` lines that look something like:

```
jamiesob: mucus.slime.com 3
tonsloye: mucus.slime.com 1
tonsloye: xboys.funnet.com.fr 3
tonsloye: warez.under.gr 1
```

Step-by-step

In this section, we will examine a complex shell programming problem and work our way through the solution.

The problem

This problem is an adaptation of the problem used in the 1997 shell programming assignment for systems administration:

Problem Definition

Your department''s FTP server provides anonymous FTP access to the **`/pub`** area of the filesystem - this area contains subdirectories (given by unit code) which contain resource materials for the various subjects offered. You suspect that this service isn't being used any more with the advent of the WWW, however, before you close this service and use the file space for something more useful, you need to prove this.

What you require is a program that will parse the FTP logfile and produce usage statistics on a given subject. This should include:

- Number of accesses per user
- Number of bytes transferred
- The number of machines which have used the area.

The program will probably be called from other scripts. It should accept (from the command line) the subject (given by the subject code) that it is to examine, followed by one or more commands. Valid commands will consist of:

- USERS - get a user and access count listing
- BYTES - bytes transmitted for the subject
- HOSTS - number of unique machines who have used the area

Background information

A cut down version of the FTP log will be examined by our program - it will consist of:

```
remote host name
file size in bytes
name of file
local username or, if guest, ID string given (anonymous FTP password)
```

For example:

```
aardvark.com   2345   /pub/85349/lectures.tar.gz    flipper@aardvark.com
138.77.8.8     112    /pub/81120/cpu.gif           sloth@topaz.cqu.edu.au
```

The FTP logfile will be called **`/var/log/ftp.log`** - we need not concern ourselves how it is produced (for those that are interested - look at **`man ftpd`** for a description of the real log file).

Anonymous FTP "usernames" are recorded as whatever the user types in as the password - while this may not be accurate, it is all we have to go on.

We can assume that all directories containing subject material branch off the **`/pub`** directory, eg.

```
/pub/85321
/pub/81120
```

Expected interaction

The following are examples of interaction with the program (**`scanlog`**):

```
[david@faile david]$ scanlog 85321 USERS
jamiesob@jasper.cqu.edu.au 1
b.spice@sworld.cqu.edu.au 22
jonesd 56

[david@faile david]$ scanlog 85321 BYTES
2322323

[david@faile david]$ scanlog 85321 HOSTS
5
```

```
[david@faile david]$ scanlog 85321 BYTES USERS
2322323
jamiesob@jasper.cqu.edu.au 1
b.spice@sworld.cqu.edu.au 22
jonesd 56
```

Solving the problem

How would you solve this problem? What would you do first?

Break it up

What does the program have to do? What are its major parts? Let's look at the functionality again - our program must:

- get a user and access count listing
- produce a the byte count on files transmitted for the subject
- list the number unique machines who have used the area and how many times

To do this, our program must first:

- Read parameters from the command line, picking out the subject we are interested in
- go through the other parameters one by one, acting on each one, calling the appropriate function
- Terminate/clean up

So, this looks like a program containing three functions. Or is it?

Look at the simple case first

It is often easier to break down a problem by walking through a simple case first.

Lets imagine that we want to get information about a subject - let''s use the code 85321. At this stage, we really don't care what the action is. What happens?

The program starts.

- We extract the first parameter from the command line. This is our subject. We might want to check if there is a first parameter - is it blank?
- Since we are only interested in this subject, we might want to go through the FTP log file and extract those entries we're interested in and keep this information in a temporary file. Our other option is to do this for every different "action" - this would probably be inefficient.
- Now, we want to go through the remaining parameters on the command line and act on each one. Maybe we should signal a error if we don't understand the action?
- At the end of our program, we should remove any temporary files we use.

Pseudo Code

If we were to pseudo code the above steps, we'd get something like:

```
# Check to see if the first parameter is blank
if  first_parameter = ""
then
  echo "No unit specified"
  exit
fi

# Find all the entries we're interested in, place this in a TEMPFILE
# Right - for every other parameter on the command line, we perform
# some

for ACTION in other_parameters
do
  # Decide if it is a valid action - act on it or give a error
done

# Remove Temp file
rm TEMPFILE


# Let's code this:
if [ "$1" = "" ]
then
  echo "No unit specified"
  exit 1
fi

# Remove $1 from the parm line

UNIT=$1
shift

# Find all the entries we're interested in
grep "/pub/$UNIT" $LOGFILE > $TEMPFILE

# Right - for every other parameter on the command line, we perform
some
for ACTION in $@
do
  process_action "$ACTION"
done

# Remove Temp file
rm $TEMPFILE
```

Ok, a few points to note:

- Notice the use of the variables **`LOGFILE`** and **`TEMPFILE`**? These would have to be defined somewhere above the code segment.
- We remove the first parameter from the command line and assign it to another variable. We do this using the shift command.
- We use **`grep`** to find all the entries in the original log file that refer to the subject we are interested in. We store these entries in a temporary file.

- The use of **$@** in the loop to process the remaining parameters is important. Why did we use it? Why not **$*** ? Hint: “1 2 3 4 5 6” isn't “1” “2” “3” “4” “5” “6” is it?
- We've invented a new function, **process_action** - we will use this to work out what to do with each action. Note that we are passing the function a parameter. Why are we enclosing it in quotes? Does it matter if we don’t? Actually, in this case, it doesn't matter if we call the function with the parameter in quotes or not, as our parameters are expected to be single words. However, what if we allowed commands like:

  ```
  find host 138.77.3.4
  ```

  If we passed this string to a function (without quotes), it would be interpreted as:

  ```
  $1=“find” $2=“host” $3=“138.77.3.4”
  ```

  This wouldn’t be entirely what we want - so, we enclose the string in quotes - producing:

  ```
  $1=“find host 138.77.3.4”
  ```

As we mentioned, in this case, we have single word commands, so it doesn’t matter, however, always try to look ahead for problems - ask yourself the figurative question - “Is my code going to work in the rain?”.

Expand function process_action

We have a function to work on - **process_action**. Again, we should pseudo code it, then implement it. Wait! We haven’t first thought about what we want it to do - always a good idea to think before you code!

This function must take a parameter, determine if it is a valid action, then perform some action on it. It is an invalid action, then we should signal an error.

Let’s try the pseudo code first:

```
process_action()
  {

    # Now, Check what we have
    case Action in
      BYTES then do a function to get bytes
      USERS then do a function to get a user list
      HOSTS then do a function to get an access count
      Something Else then echo "Unknown command $theAction"
    esac

  }
```

Right - now try the code:

```
process_action()
  {
    # Translate to upper case
    theAction=‘echo $1 | tr [a-z] [A-Z]‘

    # Now, Check what we have
    case $theAction in
      USERS) getUserList ;;
      HOSTS) getAccessCount ;;
      BYTES) getBytes ;;
      *) echo "Unknown command $theAction" ;;
    esac

  }
```

Some comments on this code:

- Note that we translate the “action command” (for example “bytes” , “users”) into upper case. This is a nicety - it just means that we’ll pick up every typing variation of the action.
- We use the case command to decide what to do with the action. We could have just as easily used a series of **`IF-THEN-ELSE-ELIF-FI`** statements - this becomes horrendous to code and read after about three conditions so case is a better option.
- As you will see, we've introduced calls to functions for each command - this again breaks to code up into bite size pieces (excuse the pun ;) to code. This follows the top-down design style.
- We will now expand each function.

Expand Function `getUserList`

Now might be a good time to revise what was required of our program - in particular, this function.

We need to produce a listing of all the people who have accessed files relating to the subject of interest and how many times they've accessed files.

Because we've separated out the entries of interest from the log file, we need no longer concern ourselves with the actual files and if they relate to the subject. We now are just interested in the users.

Reviewing the log file format:

```
aardvark.com 2345   /pub/85349/lectures.tar.gz       flipper@aardvark.com
138.77.8.8   112    /pub/81120/cpu.gif              sloth@topaz.cqu.edu.au
```

We see that user information is stored in the fourth field. If we pseudo code what we want to do, it would look something like:

```
for every_user_in the file
do
  go_through_the_file_and_count_occurences
  print this out
done
```

Expanding this a bit more, we get:

```
extract_users_from_file
for user in user_list
do
  count = 0
  while read log_file
  do
    if user = current_entry
    then
      count = count + 1
    fi
  done
  echo user count
done
```

Let's code this:

```
getUserList()
  {
    cut -f4 $TEMPFILE | sort > $TEMPFILE.users
    userList=‘uniq $TEMPFILE.users‘

    for user in $userList
    do
    {
      count=0
      while read X
      do
        if echo $X | grep $user > /dev/null
        then
          count=‘expr $count + 1‘
        fi
      done
     } < $TEMPFILE
    echo $user $count
    done

    rm $TEMPFILE.users
  }
```

Some points about this code:

- The first cut extracts a user list and places it in a temp file. A unique list of users is then created and placed into a variable.
- For every user in the list, the file is read through and each line searched for the user string. We pipe the output into **`/dev/null`**.
- If a match is made, count is incremented.
- Finally the user/count combination is printed.
- The temporary file is deleted.

Unfortunately, this code totally sucks. Why?

There are several things wrong with the code, but the most outstanding problem is the massive and useless looping being performed - the **`while`** loop reads through the file for every user. This is bad. While loops within shell scripts are very time consuming and inefficient - they are generally avoided if, as in this case, they can be. More importantly, this script doesn't make use of UNIX commands which could simplify (and speed up!) our code. Remember: don't re-invent the wheel - use existing utilities where possible.

Let's try it again, this time without the while loop:

```
getUserList()
{
  cut -f4 $TEMPFILE | sort > $TEMPFILE.users # Get user list
  userList=‘uniq $TEMPFILE.users‘

  for user in $userList                          # for every user...
  do
    count=‘grep $user $TEMPFILE.users | wc -l‘ # count how many times they are
    echo $user $count                        # in the file
  done

  rm $TEMPFILE.users
}
```

Much better! We've replaced the while loop with a simple grep command - however, there are still problems:

We don't need the temporary file

Can we wipe out a few more steps?

Next cut:

```
getUserList()
{
  userList=‘cut -f4 $TEMPFILE | sort | uniq‘

  for user in $userList
  do
    echo $user ‘grep $user $TEMPFILE | wc -l‘
  done
}
```

Beautiful!

Or is it.

What about:

```
echo ‘cut-f4 $TEMPFILE | sort | uniq -c‘
```

This does the same thing...or does it? If we didn't care what our output looked like, then this'd be ok - find out what's wrong with this code by trying it and the previous segment - compare the results. Hint: **`uniq -c`** produces a count of every *sequential* occurrence of an item in a list. What would happen if we removed the **`sort`**? How could we fix our output "problem"?

Expand Function `getAccessCount`

This function requires a the total number of unique hosts which have accessed the files. Again, as we've already separated out the entries of interest into a temporary file, we can just concentrate on the hosts field (field number one).

If we were to pseudo code this:

```
create_unique_host list
count = 0
for host in host_list
do
   count = count + 1
done
echo count
```

From the previous function, we can see that a direct translation from pseudo code to shell isn't always efficient. Could we skip a few steps and try the efficient code first? Remember - we should try to use existing UNIX commands.

How do we create a unique list? The hint is in the word unique - the **uniq** command is useful in extracting unique listings.

What are we going to use as the input to the **uniq** command? We want a list of all hosts that accessed the files - the host is stored in the first field of every line in the file. Next hint - when we see the word "field" we can immediately assume we're going to use the **cut** command. Do we have to give **cut** any parameters? In this case, no. **cut** assumes (by default) that fields are separated by tabs - in our case, this is true. However, if the delimiter was anything else, we'd have to use a "**-d**" switch, followed by the delimiter.

Next step - what about the output from **uniq**? Where does this go? We said that we wanted a count of the unique hosts - another hint - counting usually means using the **wc** command. The **wc** command (or word count command) counts characters, words and lines. If the output from the **uniq** command was one host per line, then a count of the lines would reveal the number of unique hosts.

So what do we have?

```
cut –f1
uniq
wc -l
```

Right - how do we get input and save output for each command?

A first cut approach might be:

```
cat $TEMPFILE | cut -f1 > $TEMPFILE.cut
cat $TEMPFILE.cut | uniq > $TEMPFILE.uniq
COUNT=`cat $TEMPFILE.uniq | wc -l`
echo $COUNT
```

This is very inefficient; there are several reasons for this:

- We cat a file **THREE** times to get the count. We don't even have to use cat if we really try.
- We use temp files to store results - we could use a shell variable (as in the second last line) but is there any need for this? Remember, file IO is much slower than assignments to variables, which, depending on the situation, is slower again that using pipes.
- There are four lines of code - this can be completed in **one**!

So, removing these problems, we are left with:

```
getAccessCount()
  {
    echo ‘cut -f1 $TEMPFILE | uniq | wc -l‘
  }
```

How does this work?

- The shell executes what's between **‘‘** and this is outputted by **echo**.

- This command starts with the **cut** command - a common misconception is that **cut** requires input to be piped into it - however, **cut** works just as well by accepting the name of a file to work with. The output from **cut** (a list of hosts) is piped into **uniq**.
- **uniq** then removes all duplicate host from the list - this is piped into **wc**.
- **wc** then counts the number of lines - the output is displayed.

Expand Function `getBytes`

The final function we have to write (Yes! We are nearly finished) counts the total byte count of the files that have been accessed. This is actually a fairly simple thing to do, but as you'll see, using shell scripting to do this can be very inefficient.

First, some pseudo code:

```
total = 0
while read line from file
do
  extract the byte field
  add this to the total
done

echo total
```

In shell, this looks something like:

```
getBytes()
  {
    bytes=0
    while read X
    do
      bytefield=‘echo $X | cut -f2‘
      bytes=‘expr $bytes + $bytefield‘
    done < $TEMPFILE
    echo $bytes
  }
```

...which is very inefficient (remember: looping is bad!). In this case, every iteration of the loop causes three new processes to be created, two for the first line, one for the second - creating processes takes time!

The following is a bit better:

```
getBytes()
  {
    list=‘cut -f2  $TEMPFILE ‘
    bytes=0
    for number in $list
    do
      bytes=‘expr $bytes + $number‘
    done

    echo $bytes
  }
```

The above segment of code still has looping, but is more efficient with the use of a list of values which must be added up. However, we can get smarter:

```
getBytes()
  {
   numstr=‘cut -f2 $TEMPFILE | sed "s/$/ + /g"‘
   expr $numstr 0
  }
```

Do you see what we've done? The **cut** operation produces a list of numbers, one per line. When this is piped into **sed**, the end-of-line is substituted with **" + "** - note the spaces. This is then combined into a single line string and stored in the variable **numstr**. We then get the **expr** of this string - why do we put the **0** on the end?

Two reasons:

After the **sed** operation, there is an extra **"+"** on the end - for example, if the input was:

2
3
4

The output would be:

2 +
3 +
4 +

This, when placed in a shell variable, looks like:

2 + 3 + 4 +

...which when evaluated, gives an error. Thus, placing a **0** at the end of the string matches the final **"+"** sign, and **expr** is happy

What if there wasn't a byte count? What if there were no entries - **expr** without parameters doesn't work - **expr** with **0** does.

So, is this the most efficient code?

Within the shell, yes. Probably the most efficient code would be a call to **awk** and the use of some **awk** scripting, however that is beyond the scope of this chapter and should be examined as a personal exercise.

A final note about the variables

Throughout this exercise, we've referred to **$TEMPFILE** and **$LOGFILE**. These variables should be set at the top of the shell script. **LOGFILE** refers to the location of the FTP log. **TEMPFILE** is the actual file used to store the entries of interest. This must be a unique file and should be deleted at the end of the script. It'd be an excellent idea to store this file in the **/tmp** directory (just in case your script dies and you leave the temp file laying around - **/tmp** is regularly cleaned out by the system) - it would be an even better idea to

guarantee its uniqueness by including the process ID (**$$**) somewhere within its name:

```
LOGFILE="/var/log/ftp.log"
TEMPFILE="/tmp/scanlog.$$"
```

The final program - a listing

The following is the completed shell script - notice how short the code is (think of what it would be like if we hadn't been pushing for efficiency!).

```
#!/bin/sh
#
#  FILE:        scanlog
#  PURPOSE:     Scan FTP log
#  AUTHOR:      Bruce Jamieson
#  HISTORY:     DEC 1997            Created
#
#  To do :      Truly astounding things.
#               Apart from that, process a FTP log and produce stats

#--------------------------
# globals

LOGFILE="ftp.log"
TEMPFILE="/tmp/scanlog.$$"

# functions


  #---------------------------------------
  # getAccessCount
  # - display number of unique machines that have accessed the page

  getAccessCount()
  {
    echo ‘cut -f1 $TEMPFILE  | uniq | wc -l‘
  }

  #-------------------------------------------------------
  # getUserList
  # - display the list of users who have acessed this page

  getUserList()
  {
    userList=‘cut -f4 $TEMPFILE | sort | uniq‘

    for user in $userList
    do
      echo $user ‘grep $user $TEMPFILE | wc -l‘
    done

  }

  #-------------------------------------------------------
  # getBytes
  # - calculate the amount of bytes transferred

  getBytes()
  {
```

```
   numstr=‘cut -f2 $TEMPFILE | sed "s/$/ + /g"‘
   expr $numstr 0
  }


  #-----------------------------------------------------
  # process_action
  # Based on the passed string, calls one of three functions
  #

  process_action()
  {
    # Translate to upper case
    theAction=‘echo $1 | tr [a-z] [A-Z]‘

    # Now, Check what we have
    case $theAction in
      BYTES) getBytes ;;
      USERS) getUserList ;;
      HOSTS) getAccessCount ;;
      *) echo "Unknown command $theAction" ;;
    esac

  }


#----  Main

#

if [ "$1" = "" ]
then
  echo "No unit specified"
  exit 1
fi

UNIT=$1

# Remove $1 from the parm line
shift

# Find all the entries we’re interested in
grep "/pub/$UNIT" $LOGFILE > $TEMPFILE

# Right - for every parameter on the command line, we perform some
for ACTION in $@
do
  process_action "$ACTION"
done

# Remove Temp file
rm $TEMPFILE

# We’re finished!
```

Final notes

Throughout this chapter we have examined shell programming concepts including:

- variables
- comments
- condition statements
- repeated action commands
- functions
- recursion
- traps
- efficiency, and
- structure

Be aware that different shells support different syntax - this chapter has dealt with bourne shell programming only. As a final issue, you should at some time examine the Perl programming language as it offers the full functionality of shell programming but with added, compiled-code like features - it is often useful in some of the more complex system administration tasks.

Review Questions

8.1

Write a function that equates the username in the `scanit` program with the user's full name and contact details from the `/etc/passwd` file. Modify `scanit` so its output looks something like:

```
*** Restricted Site Report ***

The following is a list of prohibited sites, users who have
visited them and on how many occasions

Bruce Jamieson x9999 mucus.slime.com 3
Elvira Tonsloy x1111 mucus.slime.com 1
Elvira Tonsloy x1111 xboys.funnet.com.fr 3
Elvira Tonsloy x1111 warez.under.gr 1
```

(Hint: the fifth field of the passwd file usually contains the full name and phone extension (sometimes))

8.2

Modify **`scanit`** so it produces a count of unique user/badsite combinations like the following:

```
*** Restricted Site Report ***

The following is a list of prohibited sites, users who have
visited them and on how many occasions

Bruce Jamieson x9999 mucus.slime.com 3
Elvira Tonsloy x1111 mucus.slime.com 1
```

```
Elvira Tonsloy x1111 xboys.funnet.com.fr 3
Elvira Tonsloy x1111 warez.under.gr 1

4 User/Site combinations detected.
```

8.3

Modify `scanit` so it produces a message something like:

```
There were no users found accessing prohibited sites!
if there were no user/badsite combinations.
```

References

Kochan S.G. et al "UNIX Shell Programming" SAMS 1993, USA

Jones, D "Shell Programming" WWW Notes

Newmarch, J "Shell Programming" http://pandonia.canberra.edu.au/OS/13_1.html

Source of `scanit`

```
#!/bin/bash
#
# AUTHOR:   Bruce Jamieson
# DATE:           Feb 1997
# PROGRAM:  scanit
# PURPOSE:  Program to analyse the output from a network
#           monitor.  "scanit" accepts a list of users to
#           and a list of "restricted" sites to compare
#           with the output from the network monitor.
#
# FILES:    scanit            shell script
#           netwatch    output from network monitor
#           netnasties  restricted site file
#
# NOTES:    This is a totally made up example - the names
#           of persons or sites used in data files are
#           not in anyway related to reality - any
#           similarity is purely coincidental :)
#
# HISTORY:  bleak and troubled :)
#

checkfile()
{
  # Goes through the netwatch file and saves user/site
  # combinations involving sites that are in the "restricted"
  # list

  while read buffer
  do
    username=‘echo $buffer | cut -d" " -f1‘
    site=‘echo $buffer | cut -d" " -f2 | sed s/\\\.//g‘
    for checksite in $badsites
    do
      checksite=‘echo $checksite | sed s/\\\.//g‘
```

```
      # echo $checksite $site
      if [ "$site" = "$checksite" ]
      then
        usersite="$username$checksite"
        if eval [ \$$usersite ]
        then
        eval $usersite=\‘expr \$$usersite + 1\‘
        else
          eval $usersite=1
        fi
      fi
    done
  done < netwatch
}

produce_report()
{
  # Goes through all possible combinations of users and
  # restricted sites - if a variable exists with the combination,
  # it is reported
  for user in $*
  do
    for checksite in $badsites
    do
      writesite=‘echo $checksite‘
      checksite=‘echo $checksite | sed s/\\\.//g‘
      usersite="$user$checksite"
      if eval [ \$$usersite ]
      then
        eval echo "$user: $writesite  \$$usersite"
        usercount=‘expr $usercount + 1‘
      fi
    done
  done
}

get_passwd_users()
{
  # Creates a user list based on the /etc/passwd file

  while read buffer
  do
    username=‘echo $buffer | cut -d":" -f1‘
    the_user_list=‘echo $username $the_user_list‘
  done  < /etc/passwd
}

check_data_files()
{
  if [ -r netwatch -a -r netnasties ]
  then
    return 0
  else
    return 1
  fi
}

# Main Program
# Uncomment the next line for debug mode
#set -x
```

```
if check_data_files
then
  echo "Datafiles found"
else
  echo "One of the datafiles missing - exiting"
  exit 1
fi

usercount=0
badsites=‘cat netnasties‘

if [ $1 ]
then
  the_user_list=$*
else
  get_passwd_users
fi
echo
echo "*** Restricted Site Report ***"
echo
echo The following is a list of prohibited sites, users who have
echo visited them and on how many occasions
echo
checkfile
produce_report $the_user_list
echo
if [ $usercount -eq 0 ]
then
  echo "There were no users found accessing prohibited sites!"
else
  echo "$usercount prohibited user/site combinations found."
fi
echo
echo

# END scanit
```

Chapter 9
Users

Introduction

Before anyone can use your system they must have an account. This chapter examines user accounts and the responsibilities of the Systems Administrators with regards to accounts. By the end of this chapter you should

- be aware of the process involved in creating and removing user accounts,
- be familiar with the configuration files that UNIX uses to store information about accounts,
- know what information you must have to create an account,
- understand the implications of choosing particular usernames, user ids and passwords,
- be aware of special accounts including the root account and the implications of using the root account,
- have been introduced to a number of public domain tools that help with account management.

Other Resources

Other material which discusses user and authentication related material includes

- Guides
  The Linux Installation and Getting Started Guide has a section (4.6) on user management. The Linux Systems Administrators Guide's chapter 9 also discusses managing user accounts. The Linux Administration Made Easy Guide also provides some discussion of account related issues in its chapter 6, General System Administration Issues.
- The RedHat 6.0 Guides also mention account management issues.

What is a UNIX account?

A UNIX account is a collection of logical characteristics that specify who the user is, what the user is allowed to do and where the user is allowed to do it. These characteristics include a

- login (or user) name,
- password,
- numeric user identifier or UID,

- a default numeric group identifier or GID,
  Many accounts belong to more than one group but all accounts have one default group.
- home directory,
- login shell,
- possibly a mail alias,
- mail file, and
- collection of startup files.

Login names

The account of every user is assigned a unique login (or user) name. The username uniquely identifies the account for people. The operating system uses the user identifier number (UID) to uniquely identify an account. The translation between UID and the username is carried out reading the `/etc/passwd` file (`/etc/passwd` is introduced below).

Login name format

On a small system, the format of login names is generally not a problem since with a small user population it is unlikely that there will be duplicates. However on a large site with hundreds or thousands of users and multiple computers, assigning a login name can be a major problem. With a larger number of users it is likely that you may get a number of people with names like David Jones, Darren Jones.

The following is a set of guidelines. They are by no means hard and fast rules but using some or all of them can make life easier for yourself as the Systems Administrator, or for your users.

- unique
  This means usernames should be unique not only on the local machine but also across different machines at the same site. A login name should identify the same person and only one person on every machine on the site. This can be very hard to achieve at a site with a large user population especially if different machines have different administrators.

  The reason for this guideline is that under certain circumstances it is possible for people with the same username to access accounts with the same username on different machines. There is an increasing trend for global logons. One username/password will get users into all of the systems they need for a given organisation. Not quite there but getting there.
- up to 8 characters
  UNIX will ignore or disallow login names that are longer. Dependent on which platform you are using.
- Lowercase
  Numbers and upper case letters can be used. Login names that are all upper

case should be avoided as some versions of UNIX can assume this to mean your terminal doesn't recognise lower case letters and every piece of text subsequently sent to your display is in uppercase.

- Easy to remember
  A random sequence of letters and numbers is hard to remember and so the user will be continually have to ask the Systems Administrator "what's my username?"
- No nicknames
  A username will probably be part of an email address. The username will be one method by which other users identify who is on the system. Not all the users may know the nicknames of certain individuals.
- A fixed format
  There should be a specified system for creating a username. Some combination of first name, last name and initials is usually the best. Setting a policy allows you to automate the procedure of adding new users. It also makes it easy for other users to work out what the username for a person might be.

Passwords

An account's password is the key that lets someone in to use the account. A password should be a secret collection of characters known only by the owner of the account.

Poor choice of passwords is the single biggest security hole on any multi-user computer system. As a Systems Administrator you should follow a strict set of guidelines for passwords (after all if someone can break the root account's password, your system is going bye, bye). In addition you should promote the use of these guidelines amongst your users.

Password guidelines

An example set of password guidelines might include

- use combinations of upper and lower case characters, numbers and punctuation characters,
- don't use random combinations of characters if they break the following two rules,
- be easy to remember,
  If a user forgets their password they can't use the system and guess whom they come and see. Also the user SHOULD NOT have to write their password down.
- be quick to type,
  One of the easiest and most used methods for breaking into a system is simply watching someone type in their password. It is harder to do if the password is typed in quickly.

- a password should be at least 6 characters long,
  The shorter a password is the easier it is to break. Some systems will not allow passwords shorter than a specified length.
- a password should not be any longer than 8 to 10 characters,
  Most systems will look as if they are accepting longer passwords but they simply ignore the extra characters. The actual size is system specific but between eight and ten characters is generally the limit.
- do not use words from ANY language,
  Passwords that are words can be cracked (you'll see how later).
- do not use combinations of just words and numbers,
  Passwords like `hello1` are just as easy to crack as `hello`.
- use combinations of words separated by punctuation characters or acronyms of uncommon phrases/song lines,
  They should be easy to remember but hard to crack. e.g. `b1gsh1p`
- change passwords regularly,
  Not too often that you forget which password is currently set.
- never reuse passwords.

The UID

Every account on a UNIX system has a unique user or login name that is used by users to identify that account. The operating system does not use this name to identify the account. Instead each account must be assigned a unique user identifier number (UID) when it is created. The UID is used by the operating system to identify the account.

UID guidelines

In choosing a UID for a new user there are a number of considerations to take into account including

- choose a UID number between 100 and 32767 (or 60000),
  Numbers between 0 and 99 are reserved by some systems for use by system accounts. Different systems will have different possible maximum values for UID numbers. Around 32000 and 64000 are common upper limits.
- UIDs for a user should be the same across machines,
  Some network systems (e.g. NFS) require that users have the same UID across all machines in the network. Otherwise they will not work properly.
- you may not want to reuse a number.
  Not a hard and fast rule. Every file is owned by a particular user id. Problems arise where a user has left and a new user has been assigned the UID of the old user. What happens when you restore from backups some files that were created by the old user? The file thinks the user with a particular UID owns it. The new user will now own those files even though the username has changed.

Home directories

Every user must be assigned a home directory. When the user logs in it is this home directory that becomes the current directory. Typically all user home directories are stored under the one directory. Many modern systems use the directory `/home`. Older versions used `/usr/users`. The names of home directories will match the username for the account.

For example, a user `jonesd` would have the home directory `/home/jonesd`

In some instances it might be decided to further divide users by placing users from different categories into different sub-directories.

For example, all staff accounts may go under `/home/staff` while students are placed under `/home/students`. These separate directories may even be on separate partitions.

Login shell

Every user account has a login shell. A login shell is simply the program that is executed every time the user logs in. Normally it is one of the standard user shells such as Bourne, `csh`, `bash` etc. However it can be any executable program.

One common method used to disable an account is to change the login shell to the program `/bin/false`. When someone logs into such an account `/bin/false` is executed and the `login:` prompt reappears.

Dot files

A number of commands, including `vi`, the mail system and a variety of shells, can be customised using dot files. A dot file is usually placed into a user's home directory and has a filename that starts with a `.` (dot). This files are examined when the command is first executed and modifies how it behaves.

Dot files are also known as `rc` files. As you should've found out by doing one of the exercises from the previous chapter `rc` is short for "run command" and is a left over from an earlier operating system.

Commands and their dot files

Table 9.1 summarises the dot files for a number of commands. The FAQs for the newsgroup `comp.unix.questions` has others.

Shell dot files

These shell dot files, particularly those executed when a shell is first executed, are responsible for

- setting up command aliases,
  Some shells (e.g. `bash`) allow the creation of aliases for various commands. A common command alias for old MS-DOS people is `dir`, usually set to mean the same as `ls -l`.
- setting values for shell variables like `PATH` and `TERM`.

Filename	Command	Explanation
~/.cshrc	/bin/csh	Executed every time C shell started.
~/.login	/bin/csh	Executed after .cshrc when logging in with C shell as the login shell.
/etc/profile	/bin/sh	Executed during the login of every user that uses the Bourne shell or its derivatives.
~/.profile	/bin/sh	Located in user's home directory. Executed whenever the user logs in when the Bourne shell is their login shell
~/.logout	/bin/csh	executed just prior to the system logging the user out (when the csh is the login shell)
~/.bash_logout	/bin/bash	executed just prior to the system logging the user out (when bash is the login shell)
~/.bash_history	/bin/bash	records the list of commands executed using the current shell
~/.forward	incoming mail	Used to forward mail to another address or a command
~/.exrc	vi	used to set options for use in vi

Table 9.1
Dot files

Skeleton directories

Normally all new users are given the same startup files. Rather than create the same files from scratch all the time, copies are usually kept in a directory called a skeleton directory. This means when you create a new account you can simply copy the startup files from the skeleton directory into the user's home directory.

The standard skeleton directory is `/etc/skel`. It should be remembered that the files in the skeleton directory are **dot** files and will not show up if you simply use `ls /etc/skel`. You will have to use the `-a` switch for `ls` to see dot files.

Exercises

9.1. Examine the contents of the skeleton directory on your system (if you have one). Write a command to copy the contents of that directory to another.
Hint: It's harder than it looks.

9.2. Use the `bash` dot files to create an alias `dir` that performs the command `ls -al`

The mail file

When someone sends mail to a user that mail message has to be stored somewhere so that it can be read. Under UNIX each user is assigned a mail file. All user mail files are placed in the same directory. When a new mail message arrives it is appended onto the end of the user's mail file.

The location of this directory can change depending on the operating system being used. Common locations are

- `/usr/spool/mail`,
- `/var/spool/mail`,
  This is the standard Linux location.
- `/usr/mail`
- `/var/mail`.

All mail in the one location

On some sites it is common for users to have accounts on a number of different computers. It is easier if all the mail for a particular user goes to the one location. This means that a user will choose one machine as their mail machine and want all their email forwarded to their account on that machine.

There are at least two ways by which mail can be forwarded

- the user can create a `.forward` file in their home directory (see Table 7.1), or
- the administrator can create an alias.

Mail aliases

If you send an e-mail message that cannot be delivered (e.g. you use the wrong address) typically the mail message will be forwarded to the `postmaster` of your machine. There is usually no account called `postmaster, postmaster` is a mail alias.

When the mail delivery program gets mail for `postmaster` it will not be able to find a matching username. Instead it will look up a specific file, under Linux `/etc/aliases`. This file will typically have an entry like

```
postmaster: root
```

This tells the delivery program that anything addressed `postmaster` should actually be delivered to the user `root`. Take a look at the /etc/aliases file on your system for other aliases.

Site aliases

Some companies will have a set policy for e-mail aliases for all staff. This means that when you add a new user you also have to update the aliases file.

For example

The Central Queensland University has aliases set up for all staff. An e-mail with an address using the format `Initial.Surname@cqu.edu.au` will be delivered to that staff member's real mail address.

In my case the alias is `d.jones@cqu.edu.au`. The main on-campus mail host has an aliases file that translates this alias into my actual e-mail address `jonesd@jasper.cqu.edu.au`.

Linux mail

The following exercise requires that you have mail delivery working on your system. You can test whether or not email is working on your system by starting one of the provided email programs (e.g. `elm`) and send yourself an email message. You do this by using only your username as the address (no @). If it isn't working, refer to the documentation from RedHat on how to get email functioning.

Exercises

9.3. Send a mail message from the `root` user to your normal user account using a mail program of your choice.

9.4. Send a mail message from the `root` user to the address `notHere`. This mail message should bounce (be unable to be delivered). You will get a returned mail message. Have a look at the mail file for postmaster. Has it increased?

9.5. Create an alias for `notHere` and try the above exercise again. If you have installed `sendmail`, the following steps should create an alias
- login as root,
- add a new line containing `notHere: root` in the file `/etc/aliases`
- run the command `newaliases`

Account configuration files

Most of the characteristics of an account mentioned above are stored in two or three configuration files. All these files are text files, each account has a one-line entry in the file with each line divided into a number of fields using colons.

Table 9.2. lists the configuration files examined and their purpose. Not all systems will have the `/etc/shadow` file. By default Linux doesn't however it is possible to install the shadow password system. On some platforms the shadow file will exist but its filename will be different.

File	Purpose
/etc/passwd	the password file, holds most of an account characteristics including username, UID, GID, GCOS information, login shell, home directory and in some cases the password
/etc/shadow	the shadow password file, a more secure mechanism for holding the password, common on more modern systems
/etc/group	the group file, holds characteristics about a system's groups including group name, GID and group members

Table 9.2
Account configuration files

/etc/passwd

/etc/passwd is the main account configuration file. Table 9.3 summarises each of the fields in the /etc/passwd file. On some systems the encrypted password will not be in the passwd file but will be in a shadow file.

Field Name	Purpose
login name	the user's login name
encrypted password *	encrypted version of the user's password
UID number	the user's unique numeric identifier
default GID	the user's default group id
GCOS information	no strict purpose, usually contains full name and address details, sometimes called the comment field
home directory	the directory in which the user is placed when they log in
login shell	the program that is run when the user logs in

** not on systems with a shadow password file*

Table 9.3
/etc/passwd

Exercises

9.6.	Examine your account's entry in the /etc/passwd field. What is your UID, GID? Where is your home directory and what is your login shell?

Everyone can read `/etc/passwd`

Every user on the system must be able to read the `/etc/passwd` file. This is because many of the programs and commands a user executes must access the information in the file. For example, when you execute the command `ls -l` command part of what the command must do is translate the UID of the file's owner into a username. The only place that information is stored is in the `/etc/passwd` file.

This is a problem

Since everyone can read the `/etc/passwd` file they can also read the encrypted password.

The problem isn't that someone might be able to decrypt the password. The method used to encrypt the passwords is supposedly a one way encryption algorithm. You aren't supposed to be able to decrypt the passwords.

Password matching

The way to break into a UNIX system is to obtain a dictionary of words and encrypt the whole dictionary. You then compare the encrypted words from the dictionary with the encrypted passwords. If you find a match you know what the password is.

Studies have shown that with a carefully chosen dictionary, between 10-20% of passwords can be cracked on any machine. Later in this chapter you'll be shown a program that can be used by the Systems Administrator to test users' passwords.

An even greater problem is the increasing speed of computers. One modern super computer is capable of performing 424,400 encryptions a second. This means that all six-character passwords can be discovered in two days and all seven-character passwords within four months.

The solution

The solution to this problem is to not store the encrypted password in the `/etc/passwd` file. Instead it should be kept in another file that only the root user can read. Remember the `passwd` program is setuid root.

This other file in which the password is stored is usually referred to as the shadow password file. It can be stored in one of a number of different locations depending on the version of UNIX you are using. A common location, and the one used by the Linux shadow password suite, is `/etc/shadow`.

During installation of Redhat 6.1 you are given the choice of using shadow passwords or not. Where possible you should use shadow passwords.

Shadow file format

Typically the shadow file consists of one line per user containing the encrypted password and some additional information including

- username,
- the date the password was last changed,
- minimum number of days before the password can be changed again,
- maximum number of days before the password must be changed,
- number of days until age warning is sent to user,
- number of days of inactivity before account should be removed,
- absolute date on which the password will expire.

The additional information is used to implement password aging. This will be discussed later in the security chapter.

Groups

A group is a logical collection of users. Users with similar needs or characteristics are usually placed into groups. A group is a collection of user accounts that can be given special permissions. Groups are often used to restrict access to certain files and programs to everyone but those within a certain collection of users.

`/etc/group`

The `/etc/group` file maintains a list of the current groups for the system and the users that belong to each group. The fields in the `/etc/group` file include

- the group name,
  A unique name for the group.
- an encrypted password (this is rarely used today) ,
- the numeric group identifier or GID, and
- the list of usernames of the group members separated by commas.

For example

On the Central Queensland University UNIX machine `jasper` only certain users are allowed to have full Internet access. All these users belong to the group called `angels`. Any program that provides Internet access has as the group owner the group `angels` and is owned by `root`. Only members of the `angels` group or the `root` user can execute these files.

The default group

Every user is the member of at least one group sometimes referred to as the default group. The default group is specified by the GID specified in the user's entry in the `/etc/passwd` file.

Since the default group is specified in `/etc/passwd` it is not necessary for the username to be added to the `/etc/group` file for the default group.

Other groups

A user can in fact be a member of several groups. Any extra groups the user is a member of are specified by entries in the `/etc/group` file.

It is not necessary to have an entry in the `/etc/group` file for the default group. However if the user belongs to any other groups they must be added to the `/etc/group` file.

Special accounts

All UNIX systems come with a number of special accounts. These accounts already exist and are there for a specific purpose. Typically these accounts will all have UIDs that are less than 100 and are used to perform a variety of administrative duties. Table 9.4. lists some of the special accounts that may exist on a machine.

Username	UID	Purpose
`root`	`0`	The super user account. Used by the Systems Administrator to perform a number of tasks. Can do anything. Not subject to any restrictions
`daemon`	`1`	Owner of many of the system daemons (programs that run in the background waiting for things to happen).
`bin`	`2`	The owner of many of the standard executable programs

Table 9.4
Special accounts

`root`

The `root` user, also known as the super user is probably the most important account on a UNIX system. This account is not subject to the normal restrictions placed on standard accounts. It is used by the Systems

Administrator to perform administrative tasks that can't be performed by a normal account.

Restricted actions

Some of the actions for which you'd use the `root` account include

- creating and modifying user accounts,
- shutting the system down,
- configuring hardware devices like network interfaces and printers,
- changing the ownership of files,
- setting and changing quotas and priorities, and
- setting the name of a machine.

Be careful

You should always be careful when logged in as `root`. When logged in as `root` you must know what every command you type is going to do. Remember the `root` account is not subject to the normal restrictions of other accounts. If you execute a command as `root` it **will** be done, whether it deletes all the files on your system or not.

The mechanics

Adding a user is a fairly mechanical task that is usually automated either through shell scripts or on many modern systems with a GUI based program. However it is still important that the Systems Administrator be aware of the steps involved in creating a new account. If you know how it works you can fix any problems which occur.

The steps to create a user include

- adding an entry for the new user to the `/etc/passwd` file,
- setting an initial password,
- adding an entry to the `/etc/group` file,
- creating the user's home directory,
- creating the user's mail file or setting a mail alias,
- creating any startup files required for the user,
- testing that the addition has worked, and
- possibly sending an introductory message to the user.

Other considerations

This chapter talks about account management which includes the mechanics of adding a new account. User management is something entirely different. When adding a new account, user management tasks that are required include

- making the user aware of the site's policies regarding computer use,
- getting the user to sign an "acceptable use" form,
- letting the user know where and how they can find information about their system, and
- possibly showing the user how to work the system.

Pre-requisite Information

Before creating a new user there is a range of information that you must know including

- the username format being used at your site,
  Are you using `djones jonesdd david jones` or perhaps you're using student or employee numbers for usernames.
- the user's name and other person information,
  Phone number, are they a computing person, someone from sales?
- where the user's home directory will be,
- will this user need a mail file on this machine or should there be an alias set up,
- startup shell,
- startup files,
- UID and GID.
  Again there should be some site wide standard for this.

Adding an `/etc/passwd` entry

For every new user, an entry has to be added to the `/etc/passwd` file. There are a variety of methods by which this is accomplished including

- using an editor,
  This is a method that is often used. However it can be unsafe and it is generally not a good idea to use it.
- the command `vipw`, or
  Some systems (usually BSD based) provide this command. `vipw` invokes an editor so the Systems Administrator can edit the passwd file safely. The command performs some additional steps that ensures that the editing is performed consistently. Some distributions of Linux supply `vipw`.
- a dedicated `adduser` program.
  Many systems, Linux included, provide a program (the name will change from system to system) that accepts a number of command-line parameters and then proceeds to perform many of the steps involved in creating a new account. The Linux command is called `adduser`

The initial password

NEVER LEAVE THE PASSWORD FIELD BLANK.

If you are not going to set a password for a user put a `*` in the password field of `/etc/passwd` or the `/etc/shadow` file. On most systems, the `*` character is considered an invalid password and it prevents anyone from using that account.

If a password is to be set for the account then the `passwd` command must be used. The user should be forced to immediately change any password set by the Systems Administrator

`/etc/group` entry

While not strictly necessary, the `/etc/group` file should be modified to include the user's login name in their default group. Also if the user is to be a member of any other group they must have an entry in the `/etc/group` file.

Editing the `/etc/group` file with an editor should be safe.

The home directory

Not only must the home directory be created but the permissions also have to be set correctly so that the user can access the directory.

The permissions of a home directory should be set such that

- the user should be the owner of the home directory,
- the group owner of the directory should be the default group that the user belongs to,
- at the very least, the owner of the directory should have `rwx` permissions, and
- the group and other permissions should be set as restrictively as possible.

The startup files

Once the home directory is created the startup files can be copied in or created. Again you should remember that this will be done as the `root` user and so `root` will own the files. You must remember to change the ownership.

For example

The following is an example set of commands that will perform these tasks.

```
mkdir home_directory
cp -pr /etc/skel/.[a-zA-Z]* home_directory
chown -R login_name home_directory
chgrp -R group_name home_directory
chmod -R 700 home_directory
```

Setting up mail

A new user will either

- want to read their mail on this machine, or
- want to read their mail on another machine.

The user's choice controls how you configure the user's mail.

A mail file

If the user is going to read their mail on this machine then you must create them a mail file. The mail file must go in a standard directory (usually `/var/spool/mail` under Linux). As with home directories it is important that the ownership and the permissions of a mail file be set correctly. The requirements are

- the user must be able to read and write the file,
  After all, the user must be able to read and delete mail messages.
- the group owner of the mail file should be the group `mail` and the group should be able to read and write to the file,
  The programs that deliver mail are owned by the group `mail`. These programs must be able to write to the file to deliver the user's mail.
- no-one else should have any access to the file,
  No-one wants anyone else peeking at their private mail.

Mail aliases and forwards

If the user's main mail account is on another machine, any mail that is sent to this machine should be forwarded to the appropriate machine. There are two methods

- a mail alias, or
- a file `~/.forward`

Both methods achieve the same result. The main difference is that the user can change the `.forward` file if they wish to. They can't modify a central alias.

Testing an account

Once the account is created, at least in some instances, you will want to test the account creation to make sure that it has worked. There are at least two methods you can use

- login as the user
- use the `su` command.

The su command

The su command is used to change from one user account to another. To a certain extent it acts like logging in as the other user. The standard format is su *username*.

```
[david@beldin david]$ su
Password:
```

Time to become the root user. su without any parameter lets you become the root user, as long as you know the password. In the following the id command is used to prove that I really have become the root user. You'll also notice that the prompt displayed by the shell has changed as well. In particular notice the # character, commonly used to indicate a shell with root permission.

```
[root@beldin david]# id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
[root@beldin david]# pwd
/home/david
```

Another point to notice is that when you don't use the "-" argument for su all that has changed is user and group ids. The current directory doesn't change.

```
[root@beldin david]# cd /
[root@beldin /]# pwd
/
[root@beldin /]# su david
[david@beldin /]$ pwd
/
[david@beldin /]$ exit
```

However, when you do use the "-" argument of the su command, it simulates a full login. This means that any startup files are executed and that the current directory becomes the home directory of the user account you "are becoming". This is equivalent to logging in as the user.

```
[root@beldin /]# su - david
[david@beldin david]$ pwd
/home/david
```

If you run su as a normal user you will have to enter the password of the user you are trying to become. If you don't specify a username you will become the root user (if you know the password).

The "-" switch

The su command is used to change from one user to another. By default, su david will change your UID and GID to that of the user david (if you know the password) but won't change much else. Using the - switch of su it is possible to simulate a full login including execution of the new user's startup scripts and changing to their home directory.

su as root

If you use the `su` command as the root user you do not have to enter the new user's password. `su` will immediately change you to the new user. `su` especially with the `-` switch is useful for testing a new account.

Exercises

9.7.	Login as yourself and perform the following steps - show your current directory (use the `pwd` command), - show you current user id and group id (use the `id` command), - use `su` to become the root user, - repeat the first two steps - use the command "`su -`" to simulate a full login as the root user, - repeat the first two steps
9.8.	What's the difference between using `su` and `su -`?

Inform the user

Lastly you should inform the user of their account details. Included in this should be some indication of where they can get assistance and some pointers on where to find more documentation.

Exercises

9.9.	By hand, create a new account for a user called David Jones.

Removing an account

Deleting an account involves reversing the steps carried out when the account was created. It is a destructive process and whenever something destructive is performed, care must always be taken. The steps that might be carried out include

- disabling the account,
- backing up and removing the associated files
- setting up mail forwards.

Situations under which you may wish to remove an account include

- as punishment for a user who has broken the rules, or
  In this situation you may only want to disable the account rather than remove it completely.
- an employee has left.

Disabling an account

Disabling an account ensures that no-one can login but doesn't delete the contents of the account. This is a minimal requirement for removing an account. There are two methods for achieving this

- change the login shell, or
  Setting the login shell to `/bin/false` will prevent logins. However it may still be possible for the user to receive mail through the account using POP mail programs like Eudora.
- change the password.

The `*` character is considered by the password system to indicate an illegal password. One method for disabling an account is to insert a `*` character into the password field. If you want to re-enable the account (with the same password) simply remove the `*`.

Another method is to simply remove the entry from the `/etc/passwd` and `/etc/shadow` files all together.

Backing up

It is possible that this user may have some files that need to be used by other people. So back everything up, just in case.

Remove the user's files

All the files owned by the account should be removed from wherever they are in the file hierarchy. It is unlikely for a user to own files that are located outside of their home directory (except for the mail file). However it is a good idea to search for them. Another use for the `find` command.

Mail for old users

On some systems, even if you delete the user's mail file, mail for that user can still accumulate on the system. If you delete an account entirely by removing it from the password field, any mail for that account will bounce.

In most cases, a user who has left will want their mail forwarded onto a new account. One solution is to create a mail alias for the user that points to their new address.

The Goals of Account Creation

As mentioned previously there is little point in adding users manually. It is a simple task which can be quite easily automated. This section looks at some of the tools you can use to automate this task.

There are at least three goals a Systems Administrator will want to achieve with adding users

- make it as simple as possible
- automate the addition of large numbers of users

- delegate the task of adding users to someone else

The following sections will show you the tools which will allow you to achieve these goals.

Making it simple

If you've completed exercise 9.8 you should by now be aware of what a straight forward, but time consuming, task creating a new user account is. Creating an account manually might be okay for one or two accounts but adding 100 this way would get quite annoying. Luckily there are a number of tools which make this process quite simple.

`useradd`

`useradd` is an executable program which significantly reduces the complexity of adding a new user. A solution to the previous exercise using `useradd` looks like this

```
useradd –c "David Jones" david
```

`useradd` will automatically create the home directory and mail file, copy files from skeleton directories and a number of other tasks. Refer to the `useradd` man page for more information.

`userdel` and `usermod`

`userdel` is the companion command to `useradd` and as the name suggests it deletes or removes a user account from the system. `usermod` allows a Systems Administrator to modify the details of an existing user account.

Graphical Tools

RedHat Linux provides a number of tools with graphical user interfaces to help both the Systems Administrator and the normal user. Tools such as `userinfo` and `userpasswd` allow normal users to modify their user accounts. RedHat also provides a command called `control-panel` which provides a graphical user interface for a number of Systems Administration related tasks including user management.

`control-panel` is in fact just a simple interface to run a number of other programs which actually perform the tasks. For example, to perform the necessary user management tasks `control-panel` will run the command `usercfg`. Diagram 9.1 provides examples of the interface provided by the `usercfg` command.

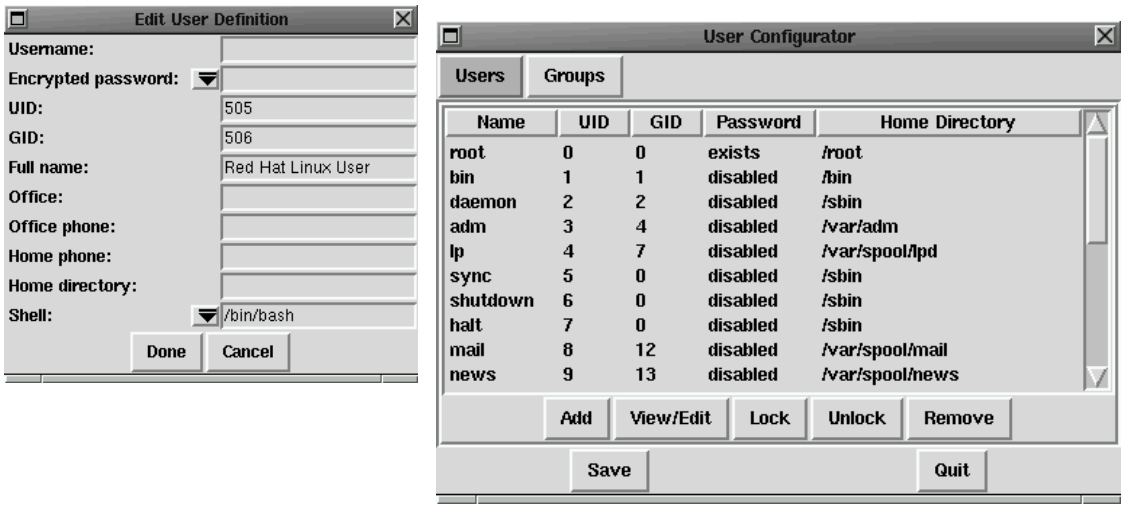


Diagram 9.1
usercfg interface

Automation

Tools with a graphical user interface are nice and simple for creating one or two users. However, when you must create hundreds of user accounts, they are a pain. In situations like this you must make use of a scripting language to automate the process.

The process of creating a user account can be divided into the following steps

- gathering the appropriate information,
- deciding on policy for usernames, passwords etc,
- creating the accounts,
- performing any additional special steps.

The steps in this process are fairly general purpose and could apply in any situation requiring the creation of a large number of user accounts, regardless of the operating system.

Gathering the information

The first part of this chapter described the type of information that is required in order to create a UNIX user account. When automating the large scale creation of user accounts this information is generally provided in an electronic format. Often this information will be extracted from a database and converted into the appropriate format.

Policy

Gathering the raw information is not sufficient. Policy must be developed which specifies rules such as username format, location of home directories,

which groups users will belong to and other information discussed earlier in the chapter.

There are no hard and fast rules for this policy. It is a case of applying whatever works best for your particular situation.

For example

CQ-PAN (http://cq-pan.cqu.edu.au) was a system managed mainly by CQU computing students. CQ-PAN provided accounts for students for a variety of reasons. During its history it has used two username formats

- ba format
  The first username format, based on that used by Freenet system, was
  `ba005 ba103 ba321`
- name format
  This was later changed to something a little more personal, firstnameLastInitialNumber. e.g. `davidj1 carolyg1`

Creating the accounts

Once you know what format the user information will be in and what formats you wish to follow for user accounts, you can start creating the accounts. Generally this will use the following steps

- read the information from the provided data file
- convert it into the format specified in the site policy
- use the UNIX account creation commands to create the accounts

Additional steps

Simply creating the accounts using the steps introduced above is usually not all that has to be done. Most sites may include additional steps in the account creation process such as

- sending an initial, welcoming email message,
  Such an email can serve a number of purposes, including informing the new users of their rights and responsibilities. It is important that users be made aware as soon as possible of what they can and can't do and what support they can expect from the Systems Administration team.
- creating email aliases or other site specific steps.

Changing passwords without interaction

Quite a few years ago there was a common problem that had to be overcome in order to automate the account creation process. This problem was how to set the new user's password without human intervention. Remember, when creating hundreds of accounts it is essential to remove all human interaction.

Given that this is such a common problem for UNIX systems, there are now a number of solutions to this problem. RedHat Linux comes with a number of solutions including the commands `chpasswd`, `newusers` and `mkpasswd`.

`mkpasswd` is an example of an Expect (http://expect.nist.gov/) script. Expect is a program that helps to automate interactive applications such as `passwd` and others including `telnet ftp` etc. This allows you to write scripts to automate processes which normally require human input.

For example

In the pre-Web days (1992), satellite weather photos were made available via FTP from a computer at James Cook University. These image files were stored using a standard filename policy which indicated which date and time the images were taken. If you wanted to view the latest weather image you had to manually `ftp` to the James Cook computer, download the latest image and then view it on your machine.

Manually ftping the files was not a large task, only 5 or 6 separate commands, however if you were doing this five times a day it got quite repetitive. Expect provides a mechanism by which a script could be written to automate this process.

Delegation

Systems Administrators are highly paid, technical staff. A business does not want Systems Administrators wasting their time performing mundane, low-level, repetitive tasks. Where possible a Systems Administrator should delegate responsibility for low-level tasks to other staff. In this section we examine one approach using the `sudo` command.

Allocating root privilege

Many of the menial tasks, like creating users and performing backups, require the access which the `root` account provides. This means that these tasks can't be allocated to junior members of staff without giving them access to everything else on the system. In most cases you don't want to do this.

There is another problem with the `root` account. If you have a number of trusted Systems Administrators the `root` account often becomes a group account. The problem with this is that since everyone knows the `root` password there is now way by which you can know who is doing what as `root`. There is no accountability. While this may not be a problem on your individual system on commercial systems it is essential to be able to track what everyone does.

sudo

A solution to these problems is the `sudo` command. `sudo` (http://www.courtesan.com/sudo/) is not a standard UNIX command but a widely available public domain tool. It comes standard on most Linux

distributions. It does not appear to be included with RedHat. You can find a copy of sudo on the 85321 Web site/CD-ROM under the Resource Materials section for week 5.

sudo allows you to allocate certain users the ability to run programs as root without giving them access to everything. For example, you might decide that the office secretary can run the adduser script, or an operator might be allowed to execute the backup script.

sudo also provides a solution to the accountability problem. sudo logs every command people perform while using it. This means that rather than using the root account as a group account, you can provide all your Systems Administrators with sudo access. When they perform their tasks with sudo, what they do will be recorded.

For example

To execute a command as root using sudo you login to your "normal" user account and then type sudo followed by the command you wish to execute. The following example shows what happens when you can and can't executable a particular command using sudo.

```
[david@mc:~]$ sudo ls
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these two things:

        #1) Respect the privacy of others.
        #2) Think before you type.

 85321.students                     archive
[david@mc:~]$ sudo cat
Sorry, user david is not allowed to execute "/bin/cat" as root on mc.
```

If the sudoers file is configured to allow you to execute this command on the current machine, you will be prompted for your normal password. You'll only be asked for the password once every five minutes.

/etc/sudoers

The sudo configuration file is usually /etc/sudoers or in some instances /usr/local/etc/sudoers. sudoers is a text file with lines of the following format

```
username  hostname=command
```

An example sudoers file might look like this

```
root  ALL=ALL
david ALL=ALL
bob   cq-pan=/usr/local/bin/backup
jo    ALL=/usr/local/bin/adduser
```

In this example the root account and the user david are allowed to execute all commands on all machines. The user bob can execute the /usr/local/bin/backup command but only on the machine cq-pan. The user jo can execute the adduser command on all machines. The sudoers man page has a more detail example and explanation.

By allowing you to specify the names of machines you can use the same `sudoers` file on all machines. This makes it easier to manage a number of machines. All you do is copy the same file to all your machines (there is a utility called `rdist` which can make this quite simple).

`sudo` advantages

`sudo` offers the following advantages

- accountability because all commands executed using `sudo` are logged, Logging on a UNIX computer, as you'll be shown in a later chapter, is done via the `syslog` system. What this means is that on a RedHat machine the information from `sudo` is logged in the file `/var/log/messages`.
- menial tasks can be allocated to junior staff without providing root access,
- using `sudo` is faster than using `su`,
- a list of users with root access is maintained,
- privileges can be revoked without changing the root password.

Some sites that use `sudo` keep the root password in an envelope in someone's draw. The root account is never used unless in emergencies where it is required.

Exercises

9.10. Install `sudo` onto your system. The source code for `sudo` is available from the Resource Materials section of the 83521 Website/CD-ROM.

9.11. Configure your version of `sudo` so that you can use it as a replacement for handing out the `root` password. What does your `/etc/sudoers` file look like?

9.12. Use `sudo` a number of times. What information is logged by the `sudo` command?

9.13. One of the listed advantages of `sudo` is the ability to log what people are doing with the `root` access. Without some extra effort this accountability can be quite pointless. Why? (Hint: the problem only really occurs with users such as `david` in the above example `sudoers` file.

Conclusions

Every user on a UNIX machine must have an account. Components of a user account can include

- login names (also called a username),
- passwords,
- the numeric user identifier or UID,
- the numeric group identifier or GID,

- a home directory,
- a login shell,
- mail aliases,
- a mail file, and
- startup files.

Configuration files related to user accounts include

- `/etc/passwd`,
- `/etc/shadow`,
- `/etc/group`, and
- to a certain extent `/etc/aliases`

Creating a user account is a mechanical task that can and often is automated. Creating an account also requires root privilege. Being the root user implies no restrictions and enables anything to be done. It is generally not a good idea to allocate this task to a junior member of staff. However, there are a number of tools which allow this and other tasks to be delegated.

Review Questions

9.1

For each of the following files/directories

- describe the purpose they fulfill
- describe the format of the file

The files are `/etc/passwd /etc/group /etc/skel`

9.2

Your company is about to fire an employee. What steps would you perform to remove the employee's account?

9.3

Set up `sudo` so that a user with the account `secretary` can run the Linux user management commands which were introduced in this chapter.

Chapter 10

Managing File Systems

Introduction

What?

In a previous chapter, we examined the overall structure of the Linux file system. This was a fairly abstract view that didn't explain how the data was physically transferred on and off the disk. Nor in fact, did it really examine the concept of "disks" or even "what" the file system "physically" existed on.

In this chapter, we shall look at how Linux interacts with physical devices (not just disks), how in particular Linux uses "devices" with respect to its file system and revisit the Linux file system - just at a lower level.

Why?

Why are you doing this? Doesn't this sound all a bit too like Operating Systems?

Unless you are content to accept that all low level interaction with the operating system occurs by a mystical form of osmosis and that you will never have to deal with:

- A Disk crash - an unfortunate physical event involving one of the read/write heads of a hard disk coming into contact with the platter (which is spinning at high speed) causing the removal of the metallic oxide (the substance that maintains magnetic polarity, thus storing data). This is usually a fatal event for the disk (and sometimes its owner).
- Adding a disk, mouse, modem terminal or a sound card - unlike some unmentionable operating systems, Linux is not "plug-and-pray". The addition of such a device requires modifications to the system.
- The accidental erasure of certain essential things called "device files" - while the accidental erasure of any file is a traumatic event, the erasure of a device file calls for special action.
- Installing or upgrading to a kernel or OS release - you may suddenly find that your system doesn't know how to talk to certain things (like your CD-ROM, your console or maybe your SCSI disk...) - you will need to find out how to solve these problems.
- Running out of some weird thing called "I-Nodes" - an event which means you can't create any more files.

... then you will definitely need to read this chapter!

Other Resources

Other material discussing file system related information includes

- HOWTOs
  CD Writing HOWTO, CDROM HOWTO, Diskless HOWTO, Jaz Drive HOWTO, Large Disk HOWTO, Multi-Disk HOWTO, Optical Disk HOWTO, Root RAID HOWTO, Software RAID HOWTO, UMSDOS HOWTO, Ext2fs Undeletion mini-HOWTO, Hard Disk Upgrade mini-HOWTO, Large Disk mini-HOWTO, Partition mini-HOWTO, Partition Rescue mini-HOWTO, Quota mini-HOWTO
- Guides
  The Linux Installation and Getting Started Guide includes a section on partitioning and preparing a disk for the installation of Linux. The Linux Systems Administrators Guide's chapter 4 provides good coverage of using disks and other storage media.
- Linux Gazette
  A free magazine distributed as part of the LDP, issue 21 of the Linux Gazette includes the article "A non-technical look inside the Ext2 file system"
- Web resources
  `http://step.polymtl.ca/~ldd/ext2fs/ext2fs_toc.html` provides a more indepth technical look at the ext2 file system. The ext2 home page is located at
  `http://web.mit.edu/tytso/www/linux/ext2.html`

A scenario

As we progress through this chapter, we will apply the information to help us solve problems associated with a very common System Administrator's task - installing a new hard disk. Our scenario is this:

Our current system has a single hard disk and it only has 10% space free (on a good day). This is causing various problems (which we will discuss during the course of this chapter) - needless to say that it is the user directories (off `/home`) that are using the most space on the system. As our IT department is very poor (we work in a university), we have been budgeting for a new hard disk for the past two years - we had bought a new one a year ago but someone drove a forklift over it. The time has finally arrived - we have a brand new 2.5 gigabyte disk (to complement our existing 500 megabyte one).

> *The size of the disk talked about here should give you some idea of how old this particular chapter is (about 3 years old). Even though today's disks are much larger the basic ideas still apply.*

How do we install it? What issues should we consider when determining its use?

Devices - Gateways to the kernel

A device is...

A device is just a generic name for any type of physical or logical system component that the operating system has to interact with (or "talk" to). Physical devices include such things as hard disks, serial devices (such as modems, mouse(s) etc.), CD-ROMs, sound cards and tape-backup drives.

Logical devices include such things as virtual terminals *[every user is allocated a terminal when they log in - this is the point at which output to the screen is sent (STDOUT) and keyboard input is taken (STDIN)]*, memory, the kernel itself and network ports.

Device files are...

Device files are special types of "files" that allow programs to interact with devices via the OS kernel. These "files" (they are not actually real files in the sense that they do not contain data) act as gateways or entry points into the kernel or kernel related "device drivers".

Device drivers are...

Device drivers are coded routines used for interacting with devices. They essentially act as the "go between" for the low level hardware and the kernel/user interface.

Device drivers may be physically compiled into the kernel (most are) or may be dynamically loaded in memory as required.

`/dev`

/dev is the location where most device files are kept. A listing of `/dev` will output the names of hundreds of files. The following is an edited extract from the **`MAKEDEV`** (a Linux program for making device files - we will examine it later) man page on some of the types of device file that exist in `/dev`:

> *Take a look at the man page for MAKEDEV on your system for an updated view of this information. Most of it will still be the same.*

- std
  Standard devices. These include mem - access to physical memory; kmem - access to kernel virtual memory;null - null device; port - access to I/O ports;
- Virtual Terminals
  This are the devices associated with the console. This is the virtual terminal tty_, where can be from 0 though 63.
- Serial Devices
  Serial ports and corresponding dialout device. For device ttyS_, there is also the device cua_ which is used to dial out with.

- Pseudo Terminals
  (Non-Physical terminals) The master pseudo-terminals are pty[p-s][0-9a-f] and the slaves are tty[p-s][0-9a-f].
- Parallel Ports
  Standard parallel ports. The devices are lp0, lp1, and lp2. These correspond to ports at 0x3bc, 0x378 and 0x278. Hence, on some machines, the first printer port may actually be lp1.
- Bus Mice
  The various bus mice devices. These include: logimouse (Logitech bus mouse), psmouse (PS/2-style mouse), msmouse (Microsoft Inport bus mouse) and atimouse (ATI XL bus mouse) and jmouse (J-mouse).
- Joystick Devices
  Joystick. Devices js0 and js1.
- Disk Devices
  Floppy disk devices. The device fd_ is the device which autodetects the format, and the additional devices are fixed format (whose size is indicated in the name). The other devices are named as fd___. The single letter _ identifies the type of floppy disk (d = 5.25" DD, h = 5.25" HD, D = 3.5" DD, H = 3.5" HD, E = 3.5" ED). The number _ represents the capacity of that format in K. Thus the standard formats are fd_d360_ fd_h1200_ fd_D720_ fd_H1440_ and fd_E2880_

  Devices fd0_ through fd3_ are floppy disks on the first controller, and devices fd4_ through fd7_ are floppy disks on the second controller.

  Hard disks. The device hdx provides access to the whole disk, with the partitions being hdx[0-20]. The four primary partitions are hdx1 through hdx4, with the logical partitions being numbered from hdx5 though hdx20. (A primary partition can be made into an extended partition, which can hold 4 logical partitions).

  Drives hda and hdb are the two on the first controller. If using the new IDE driver (rather than the old HD driver), then hdc and hdd are the two drives on the secondary controller. These devices can also be used to access IDE CD-ROMs if using the new IDE driver.

  SCSI hard disks. The partitions are similar to the IDE disks, but there is a limit of 11 logical partitions (sd_5 through sd_15). This is to allow there to be 8 SCSI disks.

  Loopback disk devices. These allow you to use a regular file as a block device. This means that images of file systems can be mounted, and used as normal. There are 8 devices, loop0 through loop7.
- Tape Devices
  SCSI tapes. These are the rewinding tape devicest_ and the non-rewinding tape device nst_.

  QIC-80 tapes. The devices are rmt8, rmt16, tape-d, and tape-reset.

Floppy driver tapes (QIC-117). There are 4 methods of access depending on the floppy tape drive. For each of access methods 0, 1, 2 and 3, the devices rft_ (rewinding) and nrft_ (non-rewinding) are created.

- CD-ROM Devices
  SCSI CD players. Sony CDU-31A CD player. Mitsumi CD player. Sony CDU-535 CD player. LMS/Philips CD player.

  Sound Blaster CD player. The kernel is capable of supporting 16 CD-ROMs, each of which is accessed as sbpcd[0-9a-f]. These are assigned in groups of 4 to each controller.

- Audio
  These are the audio devices used by the sound driver. These include mixer, sequencer, dsp, and audio.

  Devices for the PC Speaker sound driver. These are pcmixer. pxsp, and pcaudio.

- Miscellaneous
  Generic SCSI devices. The devices created are sg0 through sg7. These allow arbitrary commands to be sent to any SCSI device. This allows for querying information about the device, or controlling SCSI devices that are not one of disk, tape or CD-ROM (e.g. scanner, writable CD-ROM).

While the `/dev` directory contains the device files for many types of devices, only those devices that have device drivers present in the kernel can be used. For example, while your system may have a `/dev/sbpcd`, it doesn't mean that your kernel can support a Sound Blaster CD. To enable the support, the kernel will have to be recompiled with the Sound Blaster driver included - a process we will examine in a later chapter.

Physical characteristics of device files

If you were to examine the output of the **`ls -al`** command on a device file, you'd see something like:

```
psyche:~/sanotes$ ls -al /dev/console
crw--w--w-   1 jamiesob users      4,   0 Mar 31 09:28 /dev/console
```

In this case, we are examining the device file for the console. There are two major differences in the file listing of a device file from that of a "normal" file, for example:

```
psyche:~/sanotes$ ls -al iodev.html
-rw-r--r--   1 jamiesob users   7938 Mar 31 12:49 iodev.html
```

The first difference is the first character of the "file permissions" grouping - this is actually the file type. On directories this is a "**d**", on "normal" files it will be blank but on devices it will be "**c**" or "**b**". This character indicates **c** for

character mode or **b** for block mode. This is the way in which the device interacts - either character by character or in blocks of characters.

For example, devices like the console output (and input) character by character. However, devices like hard disks read and write in blocks. You can see an example of a block device by the following:

```
psyche:~/sanotes$ ls -al /dev/hda
brw-rw----   1 root     disk       3,   0 Apr 28  1995 /dev/hda
```

(`hda` is the first hard drive)

The second difference is the two numbers where the file size field usually is on a normal file. These two numbers (delimited by a comma) are the major and minor device numbers.

Major and minor device numbers are...

Major and minor device numbers are the way in which the kernel determines which device is being used, therefore what device driver is required. The kernel maintains a list of its available device drivers, given by the major number of a device file. When a device file is used (we will discuss this in the next section), the kernel runs the appropriate device driver, passing it the minor device number. The device driver determines which physical device is being used by the minor device number. For example:

```
psyche:~/sanotes$ ls -al /dev/hda
brw-rw----   1 root     disk       3,   0 Apr 28  1995 /dev/hda
psyche:~/sanotes$ ls -al /dev/hdb
brw-rw----   1 root     disk       3,  64 Apr 28  1995 /dev/hdb
```

What this listing shows is that a device driver, major number 3, controls both hard drives hda and hdb. When those devices are used, the device driver will know which is which (physically) because `hda` has a minor device number of 0 and `hdb` has a minor device number of 64.

Finding the devices on your system

The /proc file system provides access to certain values within the kernel of the operating system. This means you can't copy files into the /proc directory, most of the directory structure is read only. Instead you read the files under /proc to find out information about the configuration of your system and in particular the kernel.

For example, the file /proc/cpuinfo contains information about the CPU of your system.

```
[root@faile /root]# cat /proc/cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 5
model       : 8
model name  : Mobile Pentium MMX
stepping    : 1
cpu MHz           : 233.867806
fdiv_bug    : no
hlt_bug           : no
sep_bug           : no
f00f_bug    : yes
```

```
coma_bug    : no
fpu         : yes
fpu_exception     : yes
cpuid level : 1
wp          : yes
flags       : fpu vme de pse tsc msr mce cx8 mmx
bogomips    : 466.94
```

One of the other files in the proc file system is called devices. As you might expect it contains a list of the devices for which device drivers exist in your machine's kernel.

```
[root@faile /root]# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 ttyS
  5 cua
  7 vcs
 10 misc
 14 sound
 29 fb
 36 netlink
128 ptm
136 pts
162 raw
254 pcmcia

Block devices:
  1 ramdisk
  2 fd
  3 ide0
  9 md
 22 ide1
```

Many UNIX commands use the /proc file system including ps, top and uptime. The procinfo command is another useful command for displaying system status information from /proc.

```
[root@faile /root]# procinfo
Linux 2.2.12-20 (root@porky.devel.redhat.com) (gcc egcs-2.91.66) #1 1CPU
[faile]
Memory:      Total        Used        Free      Shared     Buffers
Cached
Mem:        127948      124180        3768      108864        7336
61144
Swap:        72252         128       72124

Bootup: Wed Jan 12 08:52:02 2000    Load average: 0.02 0.06 0.07 1/88 1034

user  :       0:03:05.92   5.5%  page in :    24920  disk 1:     5274r
4018w
nice  :       0:00:00.07   0.0%  page out:     6665
system:       0:00:47.27   1.4%  swap in :        1
idle  :       0:52:11.09  93.1%  swap out:       32
uptime:       0:56:04.34         context :   621650

irq  0:    336435 timer                 irq  7:         0 MSS audio codec
[0]
irq  1:     12419 keyboard              irq  8:         1 rtc
irq  2:         0 cascade [4]           irq 11:         4 i82365
irq  3:      7532 3c589_cs              irq 12:     14187 PS/2 Mouse
irq  4:         5                       irq 13:         1 fpu
```

```
irq  5:         2                      irq 14:    239393 ide0
irq  6:         3                      irq 15:     33230 ide1
```

Device files and Device drivers

The presence of a device file on your system does not mean you can actually use that device. You also need the device driver. The content of the file /proc/devices is the list of device drivers in your kernel. To use a particular device you need to have both the device driver and the device file.

Remember, the device file is simply an interface to the device driver. The device file doesn't know how to talk to the device.

For example, my laptop computer has all the device files for SCSI hard drives (/dev/sda1 /dev/sda2 etc). However, I still can't use SCSI hard-drives with the laptop because the kernel for Linux does not contain any device drivers for SCSI drives. Look at the content of the /proc/devices file in the previous example.

Why use device files?

It may seem using files is a roundabout method of accessing devices - what are the alternatives?

Other operating systems provide system calls to interact with each device. This means that each program needs to know the exact system call to talk to a particular device.

With UNIX and device files, this need is removed. With the standard open, read, write, append etc. system calls (provided by the kernel), a program may access any device (transparently) while the kernel determines what type of device it is and which device driver to use to process the call. [*You will remember from Operating Systems that system calls are the services provided by the kernel for programs.]*

Using files also allows the system administrator to set permissions on particular devices and enforce security - we will discuss this in detail later.

The most obvious advantage of using device files is shown by the way in which as a user, you can interact with them. For example, instead of writing a special program to play `.AU` sound files, you can simply:

```
psyche:~/sanotes$ cat test.au > /dev/audio
```

This command pipes the contents of the test.au file into the audio device. Two things to note: **1)** This will only work for systems with audio (sound card) support compiled into the kernel (i.e. device drivers exist for the device file) and **2)** this will only work for `.AU` files - try it with a `.WAV` and see (actually, listen) what happens. The reason for this is that `.WAV` (a Windows audio format) has to be interpreted first before it can be sent to the sound card.

> *You will not probably need to be the root user to perform the above command as the `/dev/audio` device has write permissions to all users. However, don't* ***`cat`*** *anything to a device unless you know what you are doing - we will discuss why later.*

Creating device files

There are two ways to create device files - the easy way or the hard way!

The easy way involves using the Linux command `MAKEDEV`. This is actually a script that can be found in the `/dev directory`. `MAKEDEV` accepts a number of parameters (you can check what they are in the man pages. In general, `MAKEDEV` is run as:

```
/dev/MAKEDEV device
```

where `device` is the name of a device file. If for example, you accidentally erased or corrupted your console device file (`/dev/console`) then you'd recreate it by issuing the commend:

```
/dev/MAKEDEV console
```

NOTE! This must be done as the root user

However, what if your `/dev` directory had been corrupted and you lost the `MAKEDEV` script? In this case you'd have to manually use the `mknod` command.

With the `mknod` command you must know the major and minor device number as well as the type of device (character or block). To create a device file using `mknod`, you issue the command:

```
mknod device_file_name device_type major_number minor_number
```

For example, to create the device file for COM1 a.k.a. `/dev/ttys0` (usually where the mouse is connected) you'd issue the command:

```
mknod /dev/ttyS0 c 4 240
```

Ok, so how do you know what type a device file is and what major and minor number it has so you can re-create it? The scouting (or is that the cubs?) solution to every problem in the world, *be prepared*, comes into play. Being a good system administrator, you'd have a listing of every device file stored in a file kept safely on disk. You'd issue the command:

```
ls -al /dev > /mnt/device_file_listing
```

before you lost your `/dev` directory in a cataclysmic disaster, so you could read the file and recreate the `/dev` structure (it might also be smart to copy the `MAKEDEV` script onto this same disk just to make your life easier :).

> *`MAKEDEV` is only found on Linux systems. It relies on the fact that the major and minor devices numbers for the system are hard-coded into the script - running `MAKEDEV` on a non-Linux system won't work because:*
>
> *The device names are different*
>
> *The major and minor numbers of similar devices are different*
>
> *Note however that similar scripts to `MAKEDEV` can be found on most modern versions of UNIX.*

The use and abuse of device files

Device files are used directly or indirectly in every application on a Linux system. When a user first logs in, they are assigned a particular device file for their terminal interaction. This file can be determined by issuing the command:

```
tty
```

For example:

```
psyche:~/sanotes$ tty
/dev/ttyp1
psyche:~/sanotes$ ls -al /dev/ttyp1
crw-------   1 jamiesob tty4, 193 Apr  2 21:14 /dev/ttyp1
```

Notice that as a user, I actually own the device file! This is so I can write to the device file and read from it. When I log out, it will be returned to:

```
c---------   1 root     root       4, 193 Apr  2 20:33
/dev/ttyp1
```

Try the following:

```
read X < /dev/ttyp1 ; echo "I wrote $X"
echo "hello there" > /dev/ttyp1
```

You should see something like:

```
psyche:~/sanotes$ read X < /dev/ttyp1 ; echo "I wrote $X"
hello
I wrote hello
psyche:~/sanotes$ echo "hello there" > /dev/ttyp1
hello there
```

A very important device file is that which is assigned to your hard disk. In my case `/dev/hda` is my primary hard disk, its device file looks like:

```
brw-rw----   1 root     disk       3,   0 Apr 28  1995 /dev/hda
```

Note that as a normal user, I can't directly read and write to the hard disk device file - why do you think this is?

Reading and writing to the hard disk is handled by an intermediary called the file system. We will examine the role of the file system in later sections, but for the time being, you should be aware that the file system decides how to use the disk, how to find data and where to store information about what is on the disk.

Bypassing the file system and writing directly to the device file is a very dangerous thing - device drivers have no concept of file systems, files or even the data that is stored in them; device drivers are only interested in reading and writing chunks of data (called blocks) to physical sectors of the disk. For example, by directly writing a data file to a device file, you are effectively instructing the device driver to start writing blocks of data onto the disk from where ever the disk head was sitting! This can (depending on which sector and track the disk was set to) potentially wipe out the entire file structure, boot sector and all the data. Not a good idea to try it. **NEVER** should you issue a command like:

```
cat some_file > /dev/hda1
```

As a normal user, you can't do this - but you can as root!

Reading directly from the device file is also a problem. While not physically damaging the data on the disk, by allowing users to directly read blocks, it is possible to obtain information about the system that would normally be restricted to them. For example, was someone clever enough to obtain a copy of the blocks on the disk where the shadow password file resided (a file normally protected by file permissions so users can view it), they could potentially reconstruct the file and run it through a crack program.

Exercises

10.1. Use the **`tty`** command to find out what device file you are currently logged in from. In your home directory, create a device file called `myterm` that has the same major and minor device number. Log into another session and try redirecting output from a command to `myterm`. What happens?

10.2. Use the **`tty`** command to find out what device file you are currently logged in on. Try using redirection commands to read and write directly to the device. With another user (or yourself in another session) change the permissions on the device file so that the other user can write to it (and you to theirs). Try reading and writing from each other's device files.

10.3. Log into two terminals as root. Determine the device file used by one of the sessions, take note of its major and minor device number. Delete the device file - what happens to that session. Log out of the session - now what happens? Recreate the device file.

Devices, Partitions and File systems

Device files and partitions

Apart from general device files for entire disks, individual device files for partitions exist. These are important when trying to understand how individual "parts" of a file hierarchy may be spread over several types of file system, partitions and physical devices.

Partitions are non-physical (I am deliberately avoiding the use of the word "logical" because this is a type of partition) divisions of a hard disk. IDE Hard disks may have 4 primary partitions, one of which must be a boot partition if the hard disk is the primary (modern systems have primary and secondary disk controllers) master (first hard disk) [this is the partition BIOS attempts to load a bootstrap program from at boot time].

Each primary partition can be marked as an extended partition which can be further divided into four logical partitions. By default, Linux provides device files for the four primary partitions and 4 logical partitions per primary/extended partition. For example, a listing of the device files for my primary master hard disk reveals:

```
brw-rw----   1 root     disk       3,   0 Apr 28  1995 /dev/hda
brw-rw----   1 root     disk       3,   1 Apr 28  1995 /dev/hda1
brw-rw----   1 root     disk       3,  10 Apr 28  1995 /dev/hda10
brw-rw----   1 root     disk       3,  11 Apr 28  1995 /dev/hda11
brw-rw----   1 root     disk       3,  12 Apr 28  1995 /dev/hda12
brw-rw----   1 root     disk       3,  13 Apr 28  1995 /dev/hda13
brw-rw----   1 root     disk       3,  14 Apr 28  1995 /dev/hda14
brw-rw----   1 root     disk       3,  15 Apr 28  1995 /dev/hda15
brw-rw----   1 root     disk       3,  16 Apr 28  1995 /dev/hda16
brw-rw----   1 root     disk       3,   2 Apr 28  1995 /dev/hda2
brw-rw----   1 root     disk       3,   3 Apr 28  1995 /dev/hda3
brw-rw----   1 root     disk       3,   4 Apr 28  1995 /dev/hda4
brw-rw----   1 root     disk       3,   5 Apr 28  1995 /dev/hda5
brw-rw----   1 root     disk       3,   6 Apr 28  1995 /dev/hda6
brw-rw----   1 root     disk       3,   7 Apr 28  1995 /dev/hda7
brw-rw----   1 root     disk       3,   8 Apr 28  1995 /dev/hda8
brw-rw----   1 root     disk       3,   9 Apr 28  1995 /dev/hda9
```

Partitions are usually created by using a system utility such as `fdisk`. Generally `fdisk` will ONLY be used when a new operating system is installed or a new hard disk is attached to a system.

> *Our existing hard disk would be `/dev/hda1` (we will assume that we are using an IDE drive, otherwise we'd be using SCSI devices `/dev/sd*`).*
>
> *Our new hard disk (we'll make it a slave to the first) will be `/dev/hdb1`.*

Partitions and file systems

Every partition on a hard disk has an associated file system (the file system type is actually set when `fdisk` is run and a partition is created). For example, in DOS machines, it was usual to devote the entire hard disk (therefore the entire disk contained one primary partition) to the FAT (File Allocation Table) based file system. This is generally the case for most modern operating systems including Windows 95, Win NT and OS/2.

However, there are occasions when you may wish to run multiple operating systems off the one disk; this is when a single disk will contain multiple partitions, each possibly containing a different file system.

With UNIX systems, it is normal procedure to use multiple partitions in the file system structure. It is quite possible that the file system structure is spread over multiple partitions and devices, each a different "type" of file system.

What do I mean by "type" of file system? Linux can support (or "understand", access, read and write to) many types of file systems including: minix, ext, ext2, umsdos, msdos, proc, nfs, iso9660, xenix, Sysv, coherent, hpfs.

(There is also support for the Windows 95 and Win NT file system). A file system is simply a set or rules and algorithms for accessing files. Each system is different; one file system can't read the other. Like device drivers, file systems are compiled into the kernel - only file systems compiled into the kernel can be accessed by the kernel.

To discover what file systems your system supports, you can display the contents of the `/proc/filesystems` file.

On our new disk, if we were going to use a file system that was not supported by the kernel, we would have to recompile the kernel at this point.

Partitions and Blocks

The smallest unit of information that can be read from or written to a disk is a block. Blocks can't be split up - two files can't use the same block, therefore even if a file only uses one byte of a block, it is still allocated the entire block.

When partitions are created, the first block of every partition is reserved as the boot block. However, only one partition may act as a boot partition. BIOS checks the partition table of the first hard disk at boot time to determine which partition is the boot partition. In the boot block of the boot partition there exists a small program called a bootstrap loader - this program is executed at boot time by BIOS and is used to launch the OS. Systems that contain two or more operating systems use the boot block to house small programs that ask the user to chose which OS they wish to boot. One of these programs is called lilo and is provided with Linux systems.

The second block on the partition is called the superblock. It contains all the information about the partition including information on:

- The size of the partition
- The physical address of the first data block
- The number and list of free blocks
- Information of what type of file system uses the partition
- When the partition was last modified

The remaining blocks are data blocks. Exactly how they are used and what they contain are up to the file system using the partition.

Using the partitions

So how does Linux use these partitions and file systems?

Linux logically attaches (this process is called mounting) different partitions and devices to parts of the directory structure. For example, a system may have:

```
/ mounted to /dev/hda1
/usr mounted to /dev/hda2
/home mounted to /dev/hda3
/usr/local mounted to /dev/hda4
/var/spool mounted to /dev/hdb1
/cdrom mounted to /dev/cdrom
/mnt mounted to /dev/fd0
```

Yet to a user of the system, the physical location of the different parts of the directory structure is transparent!

How does this work?

The Virtual File System

The Linux kernel contains a layer called the VFS (or Virtual File System). The VFS processes all file-oriented IO system calls. Based on the device that the operation is being performed on, the VFS decides which file system to use to further process the call.

The exact list of processes that the kernel goes through when a system call is received follows along the lines of:

- A process makes a system call.
- The VFS decides what file system is associated with the device file that the system call was made on.
- The file system uses a series of calls (called Buffer Cache Functions) to interact with the device drivers for the particular device.
- The device drivers interact with the device controllers (hardware) and the actual required processes are performed on the device.

Figure 15.1 represents this.

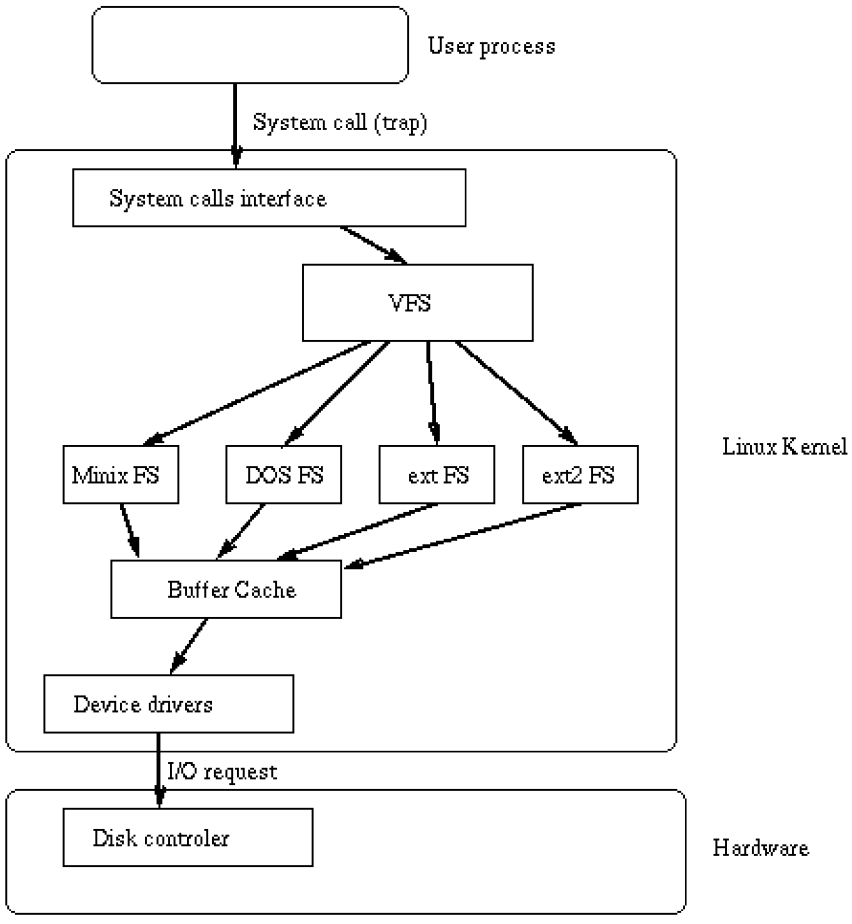


Figure 15.1
The Virtual File System

Dividing up the file hierarchy - why?

Why would you bother partitioning a disk and using different partitions for different directories?

The reasons are numerous and include:

Separation Issues

Different directory branches should be kept on different physical partitions for reasons including:

- Certain directories will contain data that will only need to be read, others will need to be both read and written. It is possible (and good practice) to mount these partitions restricting such operations.
- Directories including `/tmp` and `/var/spool` can fill up with files very quickly, especially if a process becomes unstable or the system is purposely flooded with email. This can cause problems. For example, let us assume that the `/tmp` directory is on the same partition as the `/home` directory. If the `/tmp` directory causes the partition to be filled no user will be able to write to their `/home` directory, there is no space. If `/tmp` and `/home` are on separate partitions the filling of the `/tmp` partition will not influence the `/home` directories.
- The logical division of system software, local software and home directories all lend themselves to separate partitions

Backup Issues

These include:

- Separating directories like `/usr/local` onto separate partitions makes the process of an OS upgrade easier - the new OS version can be installed over all partition except the partition that the `/usr/local` system exists on. Once installation is complete the `/usr/local` partition can be re-attached.
- The actual size of the partition can make it easier to perform backups - it isn't as easy to backup a single 2.1 Gig partition as it is to backup four 500 Meg partitions. This does depend on the backup medium you are using. Some medium will handle a 2.1 Gb partition quite easily.

Performance Issues

By spreading the file system over several partitions and devices, the IO load is spread around. It is then possible to have multiple seek operations occurring simultaneously - this will improve the speed of the system.

While splitting the directory hierarchy over multiple partitions does address the above issues, it isn't always that simple. A classic example of this is a system that contained its Web programs and data in the `/var/spool` directory. Obviously the correct location for this type of program is the `/usr` branch - probably somewhere off the `/usr/local` system. The reason for this strange

location? **ALL** the other partitions on the system were full or nearly full - this was the only place left to install the software! And the moral of the story is? When partitions are created for different branches of the file hierarchy, the future needs of the system must be considered - and even then, you won't always be able to adhere to what is "the technically correct" location to place software.

Scenario Update

At this point, we should consider how we are going to partition our new hard disk. As given by the scenario, our `/home` directory is using up a lot of space (we would find this out by using the `du` command).

We have the option of devoting the entire hard disk to the /home structure but as it is a 2.5 Gig disk we could probably afford to divide it into a couple of partitions. As the `/var/spool` directory exists on the same partition as root, we have a potential problem of our root partition filling up - it might be an idea to separate this. As to the size of the partitions? As our system has just been connected to the Internet, our users have embraced FTP - our `/home` structure is consuming 200 Megabytes but we expect this to increase by a factor of 10 over the next 2 years. Our server is also receiving increased volumes of email, so our spool directory will have to be large. A split of 2 Gigabytes to 500 Megabytes will probably be reasonable.

To create our partitions, we will use the `fdisk` program. We will create two primary partitions, one of 2 Gigabytes and one of 500 Megabytes - these we will mark as Linux partitions.

The Linux Native File System - ext2

Overview

Historically, Linux has had several native file systems. Originally there was `Minix` which supported file systems of up to 64 megabytes in size and 14 character file names. With the advent of the virtual file system (VFS) and support for multiple file systems, Linux has seen the development of `Ext FS` (Extended File System), `Xia FS` and the current `ext2 FS`.

`ext2` (the second extended file system) has longer file names (255 characters), larger file sizes (2 GB) and bigger file system support (4 TB) than any of the existing Linux file systems. In this section, we will examine how `ext2` works.

I-Nodes

`ext2` use a complex but extremely efficient method of organising block allocation to files. This system relies on data structures called **I-Nodes**. Every file on the system is allocated an I-Node - **there can never be more files than I-Nodes**.

This is something to consider when you format a partition and create the file system - you will be asked how many I-Nodes you wish create. Generally, ten percent of the file system should be I-Nodes. This figure should be increased if the partition will contain lots of small files or decreased if the partition will contain few but large files.

Figure 15.2 is a graphical representation on an I-Node.

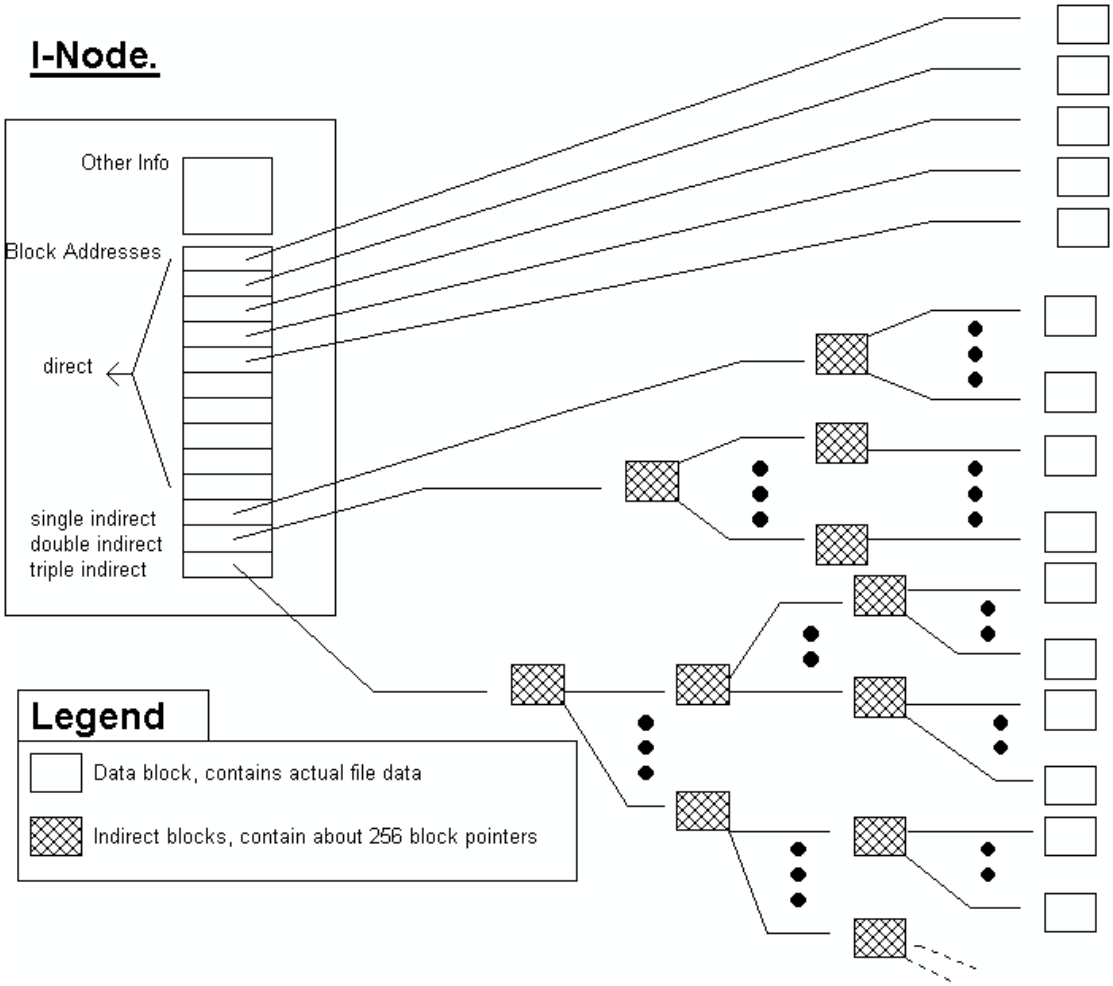


Figure 15.2
I-Node Structure

Typically an I-Node will contain:

- The owner (UID) and group owner (GID) of the file.
- The type of file - is the file a directory or another type of special file?
- User access permissions - which users can do what with the file
- The number of hard links to the file - the same physical file may be accessed under several names; we will examine how later.
- The size of the file
- The time the file was last modified

- The time the I-Node was last changed - if permissions or information on the file change then the I-Node is changed.
- The addresses of 13 data blocks - data blocks are where the contents of the file are placed.
- A single indirect pointer - this points to a special type of block called a **single indirect block**. This is a block that contains the addresses of at least 256 other data blocks; the exact number depends of the file system and implementation.
- A double indirect pointer - this points to a special type of block called a **double indirect block**. This block points to a number of single indirect blocks.
- A triple indirect pointer - this points to a special type of block called a **triple indirect block**. This block points to a number of double indirect blocks.

Using this system, `ext2` can cater for a file two gigabytes in size!

However, just because an I-Node can access all those data blocks doesn't mean that they are automatically allocated to the file when it is created - obviously! As the file grows, blocks are allocated, starting with the first direct 13 data blocks, then moving on to the single indirect blocks, then to the double, then to the triple.

Note that the actual name of the file is not stored in the I-Node. This is because the names of files are stored in directories, which are themselves files.

Physical Structure and Features

`ext2` uses a decentralised file system management scheme involving a "block group" concept. What this means is that the file systems are divided into a series of logical blocks. Each block contains a copy of critical information about the file systems (the super block and information about the file system) as well as an I-Node, and data block allocation tables and blocks. Generally, the information about a file (the I-Node) will be stored close to the data blocks. The entire system is very robust and makes file system recovery less difficult.

The `ext2` file system also has some special features which make it stand out from existing file systems including:

- Logical block size - the size of data blocks can be defined when the file system is created; this is not dependent on physical data block size.
- File system state checks - the file system keeps track of how many times it was "mounted " (or used) and what state it was left in at the last shutdown.
- The file system reserves 5% of the file system for the root user - this means that if a user program fills a partition, the partition is still useable by root (for recovery) because there is reserve space.

A more comprehensive description of the `ext2` file system can be found at `http://web.mit.edu/tytso/www/linux/ext2.html`.

dumpe2fs Command

The dumpe2fs command is useful for providing information about the super block and blocks group of a file system.

```
[root@faile /root]# dumpe2fs /dev/hda1
dumpe2fs 1.15, 18-Jul-1999 for EXT2 FS 0.5b, 95/08/09
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          459a17c0-7b5c-11d3-93ed-b8c49e459f04
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      sparse_super
Filesystem state:         not clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              6024
Block count:              24066
Reserved block count:     1203
Free blocks:              20655
Free inodes:              5999
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         2008
Inode blocks per group:   251
Last mount time:          Wed Jan 12 08:52:22 2000
Last write time:          Wed Jan 12 08:52:26 2000
Mount count:              20
Maximum mount count:      20
Last checked:             Sat Jan  1 08:35:59 2000
Check interval:           15552000 (6 months)
Next check after:         Thu Jun 29 08:35:59 2000
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:         128


Group 0: (Blocks 1 -- 8192)
  Block bitmap at 3 (+2), Inode bitmap at 4 (+3)
  Inode table at 5 (+4)
  5290 free blocks, 1983 free inodes, 2 directories
  Free blocks: 2901-2902, 2905-8192
  Free inodes: 26-2008
Group 1: (Blocks 8193 -- 16384)
  Block bitmap at 8195 (+2), Inode bitmap at 8196 (+3)
  Inode table at 8197 (+4)
  7937 free blocks, 2008 free inodes, 0 directories
  Free blocks: 8448-16384
  Free inodes: 2009-4016
Group 2: (Blocks 16385 -- 24065)
  Block bitmap at 16385 (+0), Inode bitmap at 16386 (+1)
  Inode table at 16389 (+4)
  7428 free blocks, 2008 free inodes, 0 directories
  Free blocks: 16387-16388, 16640-24065
  Free inodes: 4017-6024
```

Creating file systems

mkfs

Before a partition can be mounted (or used), it must first have a file system installed on it - with `ext2`, this is the process of creating I-Nodes and data blocks.

This process is the equivalent of formatting the partition (similar to MSDOS's "`format`" command). Under Linux, the command to create a file system is called `mkfs`.

The command is issued in the following way:

```
mkfs  [-c] [ -t fstype ]  filesys [ blocks ]
eg.
mkfs -t ext2 /dev/fd0   # Make a ext2 file system on a disk
```

where:

- `-c` forces a check for bad blocks
- `-t fstype` specifies the file system type
- `filesys` is either the device file associated with the partition or device OR is the directory where the file system is mounted (this is used to erase the old file system and create a new one)
- `blocks` specifies the number of blocks on the partition to allocate to the file system
- Be aware that creating a file system on a device with an existing file system will cause all data on the old file system to be erased.

Scenario Update

Having partitioned our disk, we must now install a file system on each partition.

`ext2` is the logical choice. Be aware that this won't always be the case and you should educate yourself on the various file systems available before making a choice.

Assuming `/dev/hdb1` is the 2GB partition and `/dev/hdb2` is the 500 MB partition, we can create ext2 file systems using the commands:

```
mkfs -t ext2 -c /dev/hdb1
mkfs -t ext2 -c /dev/hdb2
```

This assumes the default block size and the default number of I-Nodes. If we wanted to be more specific about the number of I-Nodes and block size, we could specify them. `mkfs` actually calls other programs to create the file system - in the `ext2` case, `mke2fs`. Generally, the defaults are fine - however, if we knew that we were only storing a few large files on a partition, then we'd reduce the I-Node to data block ratio. If we knew that we were storing lots of small files on a partition, we'd increase the I-Node to data block ration and

probably decrease the size of the data blocks (there is no point using 4K data blocks when the file size average is around 1K).

Exercises

10.4.	Create an ext2 file system on a floppy disk using the defaults. Create a ext2 file system on a floppy disk with only one I-node. How much disk space do you save using only 1 I-node? What restriction does this place on the floppy disk?

Mounting and UN-mounting Partitions and Devices

Mount

To attach a partition or device to part of the directory hierarchy you must mount its associated device file.

To do this, you must first have a **mount point** - this is simply a directory where the device will be attached. This directory will exist on a previously mounted device (with the exception of the root directory (/) which is a special case) and will be empty. If the directory is not empty, then the files in the directory will no longer be visible while the device to mounted to it, but will reappear after the device has been disconnected (or unmounted).

To mount a device , you use the `mount` command:

```
mount [switches] device_file mount_point
```

With some devices, mount will detect what type of file system exists on the device, however it is more usual to use mount in the form of:

```
mount [switches] -t file_system_type device_file mount_point
```

Generally, only the `root` user can use the mount command - mainly due to the fact that the device files are owned by `root`. For example, to mount the first partition on the second hard drive off the `/usr` directory and assuming it contained the `ext2` file system you'd enter the command:

```
mount -t ext2 /dev/hdb1 /usr
```

A common device that is mounted is the floppy drive. A floppy disk generally contains the `msdos` file system (but not always) and is mounted with the command:

```
mount -t msdos /dev/fd0 /mnt
```

Note that the floppy disk was mounted under the `/mnt` directory? This is because the `/mnt` directory is the usual place to temporally mount devices.

To see what devices you currently have mounted, simply type the command `mount`. Typing it on my system reveals:

```
/dev/hda3 on / type ext2 (rw)
/dev/hda1 on /dos type msdos (rw)
none on /proc type proc (rw)
```

```
/dev/cdrom on /cdrom type iso9660 (ro)
/dev/fd0 on /mnt type msdos (rw)
```

Each line tells me what device file is mounted, where it is mounted, what file system type each partition is and how it is mounted (ro = read only, rw = read/write). Note the strange entry on line three - the `proc` file system? This is a special "virtual" file system used by Linux systems to store information about the kernel, processes and current resource usages. It is actually part of the system's memory - in other words, the kernel sets aside an area of memory which it stores information about the system in - this same area is mounted onto the file system so user programs can easily gain this information.

To release a device and disconnect it from the file system, the **`umount`** command is used. It is issued in the form:

```
umount device_file
or
umount mount_point
```

For example, to release the floppy disk, you'd issue the command:

```
umount /mnt
or
umount /dev/fd0
```

Again, you must be the root user or a user with privileges to do this. You can't unmount a device/mount point that is in use by a user (the user's current working directory is within the mount point) or is in use by a process. Nor can you unmount devices/mount points which in turn have devices mounted to them.

All of this begs the question - how does the system know which devices to mount when the OS boots?

Mounting with the `/etc/fstab` file

In true UNIX fashion, there is a file which governs the behaviour of mounting devices at boot time. In Linux, this file is `/etc/fstab`. But there is a problem - if the `fstab` file lives in the `/etc` directory (a directory that will always be on the root partition (/)), how does the kernel get to the file without first mounting the root partition (to mount the root partition, you need to read the information in the `/etc/fstab` file!)? The answer to this involves understanding the kernel (a later chapter) - but in short, the system cheats! The kernel is "told" (how it is told doesn't concern us yet) on which partition to find the root file system; the kernel mounts this in read only mode, assuming the Linux native `ext2` file system, then reads the `fstab` file and re-mounts the root partition (and others) according to instructions in the file.

So what is in the file?

An example line from the `fstab` file uses the following format:

```
device_file mount_point file_system_type mount_options [n] [n]
```

The first three fields are self explanatory; the fourth field, mount_options defines how the device will be mounted (this includes information of access mode ro/rw, execute permissions and other information) - information on this can be found in the `mount` man pages (note that this field usually contains the word "defaults"). The fifth and sixth fields will usually either not be included

or be "1" - these two fields are used by the system utilities `dump` and `fsck` respectively - see the man pages for details.

As an example, the following is my `/etc/fstab` file:

```
/dev/hda5            /                    ext2    defaults        1 1
/dev/hda1            /boot                ext2    defaults        1 2
/dev/cdrom           /mnt/cdrom           iso9660 noauto,owner,ro 0 0
/dev/hda6            swap                 swap    defaults        0 0
/dev/fd0             /mnt/floppy          ext2    noauto,owner    0 0
none                 /proc                proc    defaults        0 0
none                 /dev/pts             devpts  gid=5,mode=620  0 0
```

This is a fairly standard /etc/fstab file created by installing Redhat 6.1. Linux itself has two main partitions which are used to store files, / and /boot. /boot is a small partition which contains the kernel, LILO configuration and a small number of files required to get the system going. The / file system contains every other file on my system. While this is somewhat okay for a home system it would probably be better to split this partition up based on some of the guidelines discussed in this chapter.

The swap partition is required by Linux but doesn't actually contain files. It is used directly by the kernel. You can see entries for the CD-ROM and floppy drive of the system and for mounting them under the standard /mnt directory. Lastly there are the two pseudo file systems proc and pts.

Scenario Update

The time has come for us to use our partitions. The following procedure should be followed:

Mount each partition (one at a time) off `/mnt` Eg.

```
mount -t ext2 -o defaults /dev/hdb1 /mnt
```

Copy the files from the directory that is going to reside on the partition TO the partition Eg.

```
cp - a /home /mnt
```

Modify the `/etc/fstab` file to mount the partition off the correct directory Eg.

```
/dev/hdb1  /home  ext2  defaults  1  1
```

Test your changes by rebooting and using the partition

Unmount the partition and remove the old files (or back them up).

```
umount /home
rm -r /home
mount -t ext2 -o defaults /dev/hdb1 /home
```

The new hard disk should be now installed and configured correctly!

Exercises

10.5.	Mount a floppy disk under the `/mnt/floppy` directory.
10.6.	Carefully examine your `/etc/fstab file` - work out what each entry means.

10.7.	Change to the `/mnt/floppy` directory (while the disk is mounted) - now try to unmount the disk - does this work? Why/Why not?

File Operations

Creating a file

When a file is created, the following process is performed:

- An I-Node is allocated to the file.
  (If one is available)
- An entry is added to the current directory - remember, the directory is a file itself. This entry contains the name of the file and a pointer to I-Node used by the file. The link count on the file's I-Node is set to 1 (any I-Node with a link count of 0 is not in use).
- Any blocks required to store the file contents are allocated.

Linking files

As we have previously encountered, there are occasions when you will want to access a file from several locations or by several names. The process of doing this is called linking.

There are two methods of doing this - **Hard Linking** and **Soft Linking**.

Hard Links are generated by the following process:

- An entry is added to the current directory with the name of the link together with a pointer to the I-Node used by the original file.
- The I-Node of the original file is updated and the number of files linked to it is incremented.

Soft Links are generated by the following process:

- An I-Node is allocated to the soft link file - the type of file is set to soft-link.
- An entry is added to the current directory with the name of the link together with a pointer to the allocated I-Node.
- A data block is allocated for the link in which is placed the name of the original file.

Programs accessing a soft link cause the file system to examine the location of the original (linked-to) file and then carry out operations on that file. The following should be noted about links:

- Hard links may only be performed between files on the same physical partition - the reason for this is that I-Nodes pointers can only point to I-Nodes of the same partition
- Any operation performed on the data in link is performed on the original file.
- Any `chmod` operations performed on a hard link are reflected on both the hard link file and the file it is linked to. `chmod` operations on soft links are

reflected on the original file but not on the soft link - the soft link will always have full file permissions (lrwxrwxrwx) .

So how do you perform these mysterious links?

ln

The command for both hard and soft link files is ln. It is executed in the following way:

```
ln source_file link_file_name   # Hard Links
or
ln -s source_file link_file_name# Soft Links
```

For example, look at the following operations on links:

Create the file and check the `ls` listing:

```
psyche:~$ touch base
psyche:~$ ls -al base
-rw-r--r--   1 jamiesob users   0 Apr  5 17:09 base
```

Create a soft link and check the `ls` listing of it and the original file

```
psyche:~$ ln -s base softbase
psyche:~$ ls -al softbase
lrwxrwxrwx   1 jamiesob users   4 Apr  5 17:09 softbase -> base
psyche:~$ ls -al base
-rw-r--r--   1 jamiesob users   0 Apr  5 17:09 base
```

Create a hard link and check the `ls` listing of it, the soft link and the original file

```
psyche:~$ ln base hardbase
psyche:~$ ls -al hardbase
-rw-r--r--   2 jamiesob users   0 Apr  5 17:09 hardbase
psyche:~$ ls -al base
-rw-r--r--   2 jamiesob users   0 Apr  5 17:09 base
psyche:~$ ls -il base
132307 -rw-r--r--   2 jamiesob users   0 Apr  5 17:09 base
psyche:~$ ls -il softbase
132308 lrwxrwxrwx   1 jamiesob users   4 Apr  5 17:09 softbase ->base
psyche:~$ ls -il hardbase
132307 -rw-r--r--   2 jamiesob users   0 Apr  5 17:09 hardbase
```

Note the last three operations (checking the I-Node number) - see how the hard link shares the I-Node of the original file? Links are removed by simply deleting the link with the rm (or on non-Linux systems `unlink`) command. Note that deleting a file that has soft links is different to deleting a file with hard links - deleting a soft-linked file causes the I-Node (thus data blocks) to be deallocated - no provision is made for the soft link which is now "pointing" to a file that doesn't exist.

However, a file with hard links to it has its entry removed from the directory, but neither its I-Node nor data blocks are deallocated - the link count on the I-Node is simply decremented. The I-Node and data blocks will only be deallocated when there are no other files hard linked to it.

Exercises

10.8.	Locate all files on the system that are soft links (Hint: use `find`).

Checking the file system

Why Me?

It is a sad truism that anything that can go wrong will go wrong - especially if you don't have backups! In any event, file system "crashes" or problems are an inevitable fact of life for a System Administrator.

Crashes of a non-physical nature (i.e. the file system becomes corrupted) are non-fatal events - there are things a system administrator can do before issuing the last rites and restoring from one of their copious backups :)

You will be informed of the fact that a file system is corrupted by a harmless, but feared little messages at boot time, something like:

```
Can't mount /dev/hda1
```

If you are lucky, the system will ignore the file system problems and try to mount the corrupted partition READ ONLY.

It is at this point that most people enter a hyperactive frenzy of swearing, violent screaming tantrums and self-destructive cranial impact diversions (head butting the wall).

What to do

It is important to establish that the problem is logical, not physical. There is little you can do if a disk head has crashed (on the therapeutic side, taking the offending hard disk into the car park and beating it with a stick can produce favourable results). A logical crash is something that is caused by the file system becoming confused. Things like:

- Many files using the one data block.
- Blocks marked as free but being used and vice versa.
- Incorrect link counts on I-Nodes.
- Differences in the "size of file" field in the I-Node and the number of data blocks actually used.
- Illegal blocks within files.
- I-Nodes contain information but are not in any directory entry (these type of files, when recovered, are placed in the `lost+found` directory).
- Directory entries that point to illegal or unallocated I-Nodes.

are the product of file system confusion. These problems will be detected and (usually) fixed by a program called fsck.

`fsck`

`fsck` is actually run at boot time on most Linux systems. Every x number of boots, `fsck` will do a comprehensive file system check. In most cases, these boot time runs of `fsck` automatically fix problems - though occasionally you may be prompted to confirm some `fsck` action. If however, `fsck` reports

some drastic problem at boot time, you will usually be thrown in to the `root` account and issued a message like:

```
************************************
fsck returned error code - REBOOT NOW!
************************************
```

It is probably a good idea to manually run `fsck` on the offending device at this point (we will get onto how in a minute).

At worst, you will get a message saying that the system can't mount the file system at all and you have to reboot. It is at this point you should drag out your rescue disks (which of course contain a copy of `fsck`) and reboot using them. The reason for booting from an alternate source (with its own file system) is because it is quite possible that the location of the `fsck` program (`/sbin`) has become corrupted as has the `fsck` binary itself! It is also a good idea to run `fsck` only on unmounted file systems.

Using `fsck`

`fsck` is run by issuing the command:

```
fsck file_system
```

where `file_system` is a device or directory from which a device is mounted.

`fsck` will do a check on all I-Nodes, blocks and directory entries. If it encounters a problem to be fixed, it will prompt you with a message. If the message asks if `fsck` can SALVAGE, FIX, CONTINUE, RECONNECT or ADJUST, then it is usually safe to let it. Requests involving REMOVE and CLEAR should be treated with more caution.

What caused the problem?

Problems with the file system are caused by:

- People turning off the power on a machine without going through the shutdown process - this is because Linux uses a very smart READ and WRITE disk cache - this cache is only flushed (or written to disk) periodically and on shutdown. `fsck` will usually fix these problems at the next boot.
- Program crashes - problems usually occur when a program is using several files and suddenly crashes without closing them. `fsck` usually easily fixes these problems.
- Kernel and system crashes - the kernel may become unstable (especially if you are using new, experimental kernels) and crash the system. Depending on the circumstances, the file system will usually be recoverable.

Exercises

10.9.	Mount the disk created in an earlier exercise. Copy the contents of your home directory to the disk. Now copy the kernel to it (`/vmlinuz`) but during the copy eject the disk (the idea is to do this while the light which indicates writing to disk is on). Now run `fsck` on that disk.

Conclusion

Having read and absorbed this chapter you will be aware that:

- Linux supports many file systems.
- The process of using many file systems, partitions and devices acting in concert to produce a directory structure allows for greater flexibility, performance and system integrity.
- The implementation of this process requires a number of components working in conjunction including: device drivers, device files, the virtual file system, specific file systems and user commands.

Review questions

10.1

As a System Administrator, you have been asked to set up a new system. The system will contain two hard disks, each 2.5 Gb in size. What issues must you consider when installing these disks? What questions should you be asking about the usage of the disks?

10.2

You have noticed that at boot time, not all the normal messages are appearing on the screen. You have also discovered that X-Windows won't run. Suggest possible reasons for this and the solutions to the problems.

10.3

A new hard disk has been added to your system to store the print spool in. List all the steps in adding this hard disk to the system.

10.4

You have just dropped your Linux box while it was running (power was lost during the system's short flight) - the system boots but will not mount the hard disk. Discuss possible reasons for the problem and the solutions.

10.5

What are links used for? What are the differences between hard and soft links?

Chapter 11

Backups

Like most of those who study history, he (Napoleon III) learned from the mistakes of the past how to make new ones.

A.J.P. Taylor.

Introduction

This is THE MOST IMPORTANT responsibility of the System Administrator. Backups MUST be made of all the data on the system. It is inevitable that equipment will fail and that users will "accidentally" delete files. There should be a safety net so that important information can be recovered.

It isn't just users who accidentally delete files

A friend of mine who was once the administrator of a UNIX machine (and shall remain nameless, but is now a respected Academic at CQU) committed one of the great no-no's of UNIX Administration.

Early on in his career he was carefully removing numerous old files for some obscure reason when he entered commands resembling the following (he was logged in as `root` when doing this).

```
cd / usr/user/panea           notice the mistake
rm -r *
```

The first command contained a typing mistake (the extra space) that meant that instead of being in the directory `/usr/user/panea` he was now in the `/` directory. The second command says delete everything in the current directory and any directories below it. Result: a great many files removed.

The moral of this story is that everyone makes mistakes. Root users, normal users, hardware and software all make mistakes, break down or have faults. This means you must keep backups of any system.

Other Resources

Other resources which discuss backups and related information include

- How-tos
  Linux ADSM Mini-Howto,
- The LAME guide's chapter on backup and restore procedures
- The Linux Systems Administrators Guide's chapter (10) on backups

Backups aren't enough

Making sure that backups are made at your site isn't enough. Backups aren't any good if you can't restore the information contained. You must have some sort of plan to recover the data. That plan should take into account all sorts of scenarios. Recovery planning is not covered to any great extent in this text. That doesn't mean it isn't important.

Characteristics of a good backup strategy

Backup strategies change from site to site. What works on one machine may not be possible on another. There is no standard backup strategy. There are however a number of characteristics that need to be considered including

- ease of use,
- time efficiency,
- ease of restoring files,
- ability to verify backups,
- tolerance of faulty media, and
- portabilty to a range of machines.

Ease of use

If backups are easy to use, you will use them. AUTOMATE!! It should be as easy as placing a tape in a drive, typing a command and waiting for it to complete. In fact you probably shouldn't have to enter the command, it should be automatically run.

When backups are too much work

At many large computing sites operators are employed to perform low-level tasks like looking after backups. Looking after backups generally involves obtaining a blank tape, labelling it, placing it in the tape drive, waiting for the information to be stored on the tape and then storing it away.

A true story that is told by an experienced Systems Administrator is about an operator that thought backups took too long to perform. To solve this problem the operator decided backups finished much quicker if you didn't bother putting the tape in the tape drive. You just labelled the blank tape and placed it in storage.

Quite alright as long as you don't want to retrieve anything from the backups.

Time efficiency

Obtain a balance to minimise the amount of operator, real and CPU time taken to carry out the backup and to restore files. The typical tradeoff is that a quick backup implies a longer time to restore files. Keep in mind that you will in general perform more backups than restores.

On some large sites, particular backup strategies fail because there aren't enough hours in a day. Backups scheduled to occur every 24 hours fail because the previous backup still hasn't finished. This obviously occurs at sites which have large disks.

Ease of restoring files

The reason for doing backups is so you can get information back. You will have to be able to restore information ranging from a single file to an entire file system. You need to know on which media the required file is and you need to be able to get to it quickly.

This means that you will need to maintain a table of contents and label media carefully.

Ability to verify backups

YOU MUST VERIFY YOUR BACKUPS. The safest method is once the backup is complete, read the information back from the media and compare it with the information stored on the disk. If it isn't the same then the backup is not correct.

Well that is a nice theory but it rarely works in practice. This method is only valid if the information on the disk hasn't changed since the backup started. This means the file system cannot be used by users while a backup is being performed or during the verification. Keeping a file system unused for this amount of time is not often an option.

Other quicker methods include

- restoring a random selection of files from the start, middle and end of the backup,
  If these particular files are retrieved correctly the assumption is that all of the files are valid.
- create a table of contents during the backup; afterwards read the contents of the tape and compare the two.

These methods also do not always work. Under some conditions and with some commands the two methods will not guarantee that your backup is correct.

Tolerance of faulty media

A backup strategy should be able to handle

- faults in the media, and

- physical dangers.

There are situations where it is important that

- there exist at least two copies of full backups of a system, and
- that at least one set should be stored at another site.

Consider the following situation.

A site has one set of full backups stored on tapes. They are currently performing another full backup of the system onto the same tapes. What happens when the backup system is happily churning away when it gets about halfway and crashes (the power goes off, the tape drive fails etc). This could result in the both the tape and the disk drive being corrupted. Always maintain duplicate copies of full backups.

An example of the importance of storing backups off site was the Pauls ice-cream factory in Brisbane. The factory is located right on the riverbank and during the early 1970's Brisbane suffered problems caused by a major flood. The Pauls' computer room was in the basement of their factory and was completely washed out. All the backups were kept in the computer room.

Portabilty to a range of platforms

There may be situations where the data stored on backups must be retrieved onto a different type of machine. The ability for backups to be portable to different types of machine is often an important characteristic.

For example:

The computer currently being used by a company is the last in its line. The manufacturer is bankrupt and no one else uses the machine. Due to unforeseen circumstances the machine burns to the ground. The Systems Administrator has recent backups available and they contain essential data for this business. How are the backups to be used to reconstruct the system?

Considerations for a backup strategy

Apart from the above characteristics, factors that may affect the type of backup strategy implemented will include

- the available commands
  The characteristics of the available commands limit what can be done.
- available hardware
  The capacity of the backup media to be used also limits how backups are performed. In particular how much information can the media hold?
- maximum expected size of file systems
  The amount of information required to be backed up and whether or not the combination of the available software and hardware can handle it. A suggestion is that individual file systems should never contain more information than can fit easily onto the backup media.

- importance of the data
  The more important the data is, the more important that it be backed up regularly and safely.
- level of data modification
  The more data being created and modified, the more often it should be backed up. For example the directories `/bin` and `/usr/bin` will hardly ever change so they rarely need backing up. On the other hand directories under `/home` are likely to change drastically every day.

The components of backups

There are basically three components to a backup strategy. The

- scheduler
  Decides when the backup is performed.
- transport, and
  The command that moves the backup from the disks to the backup media.
- media
  The actual physical device on which the backup is stored.

Scheduler

The scheduler is the component that decides when backups should be performed and how much should be backed up. The scheduler could be the `root` user or a program, usually `cron` (discussed in a later chapter).

The amount of information that the scheduler backs up can have the following categories

- full backups,
  All the information on the entire system is backed up. This is the safest type but also the most expensive in machine and operator time and the amount of media required.
- partial backups, or
  Only the busier and more important file systems are backed up. One example of a partial backup might include configuration files (like `/etc/passwd`), user home directories and the mail and news spool directories. The reasoning is that these files change the most and are the most important to keep a track of. In most instances this can still take substantial resources to perform.
- incremental backups.
  Only those files that have been modified since the last backup are backed up. This method requires less resources but a large amount of incremental backups make it more difficult to locate the version of a particular file you may desire.

Transport

The transport is a program that is responsible for placing the backed-up data onto the media. There are quite a number of different programs that can be used as transports. Some of the standard UNIX transport programs are examined later in this chapter.

There are two basic mechanisms that are used by transport programs to obtain the information from the disk

- image, and
- through the file system.

Image transports

An image transport program bypasses the file system and reads the information straight off the disk using the raw device file. To do, this the transport program needs to understand how the information is structured on the disk. This means that transport programs are linked very closely to exact file systems since different file systems structure information differently.

Once read off the disk, the data is written byte by byte from disk onto tape. This method generally means that backups are usually quicker than the "file by file" method. However restoration of individual files generally takes much more time.

Transport programs that use the method include `dd`, `volcopy` and `dump`.

File by file

Commands performing backups using this method use the system calls provided by the operating system to read the information. Since almost any UNIX system uses the same system calls, a transport program that uses the file by file method (and the data it saves) is more portable.

File by file backups generally take more time but it is generally easier to restore individual files. Commands that use this method include `tar` and `cpio`.

Backing up FAT and EXT2 file systems

If you are like most people using this text then chances are that your Linux computer contains both FAT and EXT2 file systems. The FAT file systems will be used by the version of Windows you were originally running while the EXT2 file systems will be those used by Linux.

Of course being the trainee computing professional you are backups of your personal computer are performed regularly. It would probably be useful to you to be able to backup both the FAT and EXT2 file systems at the same time, without having to switch operating systems.

Well doing this from Windows isn't going to work. Windows still doesn't read the EXT2 file system. So you will have to do it from Linux. Which type of transport do you use for this, image or file by file?

Well here's a little excerpt from the man page for the dump command, the one of the image transports available on Linux.

```
It might be considered a bug that this version of dump can only
handle ext2 filesystems.  Specifically, it does not work with FAT
filesystems.
```

If you think about it this shortcoming is kind of obvious.

The dump command does not use the kernel file system code. It is an image transport. This means it must know everything about the filesystem it is going to backup. How are directories structured, how are the data blocks for files store on the system, how is file metadata (e.g. permissions, file owners etc) stored and many more questions.

The people who wrote dump included this information into the command.

They didn't include any information about the FAT file system. So dump can't backup FAT file systems.

File by file transports on the other hand can quite happily backup any file system which you can mount on a Linux machine. In this situation the virtual file system takes care of all the differences and all the file-by-file transport knows about are what appear to be normal Linux files.

Media

Backups are usually made to tape based media. There are different types of tape. Tape media can differ in

- physical size and shape, and
- amount of information that can be stored.
  From 100Mb up to 8Gb.

Different types of media can also be more reliable and efficient. The most common type of backup media used today are 4 millimetre DAT tapes.

Reading

Under the Resource Materials section for Week 6 on the 85321 Web site/CD-ROM you will find a pointer to the USAIL resources on backups. This includes a pointer to discussion about the different type of media which are available.

Commands

As with most things, the different versions of UNIX provide a plethora of commands that could possibly act as the transport in a backup system. The following table provides a summary of the characteristics of the more common programs that are used for this purpose.

Command	Availability	Characteristics
`dump/restore`	BSD systems	image backup, allows multiple volumes, not included on most AT&T systems
`tar`	almost all systems	file by file, most versions do not support multiple volumes, intolerant of errors
`cpio`	AT&T systems	file by file, can support multiple volumes some versions don't,

Table 11.1.
The Different Backup Commands.

There are a number of other public domain and commercial backup utilities available which are not listed here.

`dump` and `restore`

A favourite amongst many Systems Administrators, `dump` is used to perform backups and `restore` is used to retrieve information from the backups.

These programs are of BSD UNIX origin and have not made the jump across to SysV systems. Most SysV systems do not come with `dump` and `restore`. The main reason is that since `dump` and `restore` bypass the file system, they must know how the particular file system is structured. So you simply can't recompile a version of `dump` from one machine onto another (unless they use the same file system structure).

Many recent versions of systems based on SVR4 (the latest version of System V UNIX) come with versions of `dump` and `restore`.

`dump` on Linux

There is a version of `dump` for Linux. However, it may be possible that you do not have it installed on your system. RedHat Linux does include an RPM package which contains `dump`. If your system doesn't have `dump` and `restore` installed you should install it now. RedHat provides a couple of tools to installe these packages: `rpm` and `glint`. `glint` is the GUI tool for managing packages. Refer to the RedHat documentation for more details on using these tools.

`dump`

The command line format for `dump` is

```
dump [ options [ arguments ] ] file system
dump [ options [ arguments ] ] filename
```

Arguments must appear after all options and must appear in a set order.

`dump` is generally used to backup an entire partition (file system). If given a list of filenames, `dump` will backup the individual files.

`dump` works on the concept of levels (it uses 9 levels). A dump level of 0 means that all files will be backed up. A dump level of 1...9 means that all files that have changed since the last dump of a lower level will be backed up. Table 11.2 shows the arguments for `dump`.

Options	Purpose
`0-9`	dump level
`a` *`archive-file`*	archive-file will be a table of contents of the archive.
`f` *`dump-file`*	specify the file (usually a device file) to write the dump to, a – specifies standard output
`u`	update the dump record (`/etc/dumpdates`)
`v`	after writing each volume, rewind the tape and verify. The file system must not be used during dump or the verification.

Table 11.2. Arguments for `dump`

There are other options. Refer to the man page for the system for more information.

For example:

```
dump 0dsbfu 54000 6000 126 /dev/rst2 /usr
```

full backup of `/usr` file system on a 2.3 Gig 8mm tape connected to device `rst2` The numbers here are special information about the tape drive the backup is being written on.

The `restore` command

The purpose of the `restore` command is to extract files archived using the `dump` command. `restore` provides the ability to extract single individual files, directories and their contents and even an entire file system.

```
restore -irRtx [ modifiers ] [ filenames ]
```

The `restore` command has an interactive mode where commands like `ls` etc can be used to search through the backup.

Arguments	Purpose
i	interactive, directory information is read from the tape after which you can browse through the directory hierarchy and select files to be extracted.
r	restore the entire tape. Should only be used to restore an entire file system or to restore an incremental tape after a full level 0 restore.
t	table of contents, if no filename provided, root directory is listed including all subdirectories (unless the h modifier is in effect)
x	extract named files. If a directory is specified, it and all its sub-directories are extracted.

Table 11.3.
Arguments for the restore Command.

Modifiers	Purpose
a *archive-file*	use an archive file to search for a file's location. Convert contents of the dump tape to the new file system format
d	turn on debugging
h	prevent hierarchical restoration of sub-directories
v	verbose mode
f *dump-file*	specify *dump-file* to use, - refers to standard input
s *n*	skip to the nth dump file on the tape

Table 11.4.
Argument modifiers for the restore Command.

Using dump and restore without a tape

Not many of you will have tape drives or similar backup media connected to your Linux machine. However, it is important that you experiment with the dump and restore commands to gain an understanding of how they work. This section offers a little kludge which will allow you to use these commands without a tape drive. The method relies on the fact that UNIX accesses devices through files.

Our practice file system

For all our experimentation with the commands in this chapter we are going to work with a practice file system. Practising backups with hard-drive partitions

is not going to be all that efficient as they will almost certainly be very large. Instead we are going to work with a floppy drive.

The first step then is to format a floppy with the `ext2` file system. By now you should know how to do this. Here's what I did to format a floppy and put some material on it.

```
[root@beldin]# /sbin/mke2fs /dev/fd0
mke2fs 1.10, 24-Apr-97 for EXT2 FS 0.5b, 95/08/09
Linux ext2 filesystem format
Filesystem label=
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done
[root@beldin]# mount -t ext2 /dev/fd0 /mnt/floppy
[root@beldin]# cp /etc/passwd /etc/issue /etc/group /var/log/messages
/mnt/floppy
[root@beldin dump-0.3]#
```

Doing a level 0 dump

So I've copied some important stuff to this disk. Let's assume I want to do a level 0 dump of the `/mnt/floppy` file system. How do I do it?

```
[root@beldin]# /sbin/dump 0f /tmp/backup /mnt/floppy
  DUMP: Date of this level 0 dump: Sun Jan 25 15:05:11 1998
  DUMP: Date of last level 0 dump: the epoch
  DUMP: Dumping /dev/fd0 (/mnt/floppy) to /tmp/backup
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 42 tape blocks on 0.00 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: DUMP: 29 tape blocks on 1 volumes(s)
  DUMP: Closing /tmp/backup
  DUMP: DUMP IS DONE
```

The arguments to the `dump` command are

- `0`
  This tells `dump` I wish to perform a level 0 dump of the file system.
- `f`
  This is telling dump that I will tell it the name of the file that it should write the backup to.
- `/tmp/backup`
  This is the name of the file I want the backup to go to. Normally, this would be the device file for a tape drive or other backup device. However, since I don't have one I'm telling it a normal file.
- `/mnt/floppy`
  This is the file system I want to backup.

What this means is that I have now created a file, `/tmp/backup`, which contains a level 0 dump of the floppy.

```
[root@beldin]# ls -l /tmp/backup
-rw-rw-r--   1 root     tty         20480 Jan 25 15:05 /tmp/backup
```

Restoring the backup

Now that we have a dump archive to work with, we can try using the `restore` command to retrieve files.

```
[root@beldin dump-0.3]# /sbin/restore -if /tmp/backup
restore > ?
Available commands are:
        ls [arg] - list directory
        cd arg - change directory
        pwd - print current directory
        add [arg] - add ‘arg’ to list of files to be extracted
        delete [arg] - delete ‘arg’ from list of files to be extracted
        extract - extract requested files
        setmodes - set modes of requested directories
        quit - immediately exit program
        what - list dump header information
        verbose - toggle verbose flag (useful with ‘‘ls’’)
        help or ‘?’ - print this list
If no ‘arg’ is supplied, the current directory is used
restore > ls
.:
group       issue       lost+found/ messages    passwd

restore > add passwd
restore > extract
You have not read any tapes yet.
Unless you know which volume your file(s) are on you should start
with the last volume and work towards towards the first.
 Specify next volume #: 1
Mount tape volume 1
Enter ‘‘none’’ if there are no more tapes
otherwise enter tape name (default: /tmp/backup)
set owner/mode for ’.’? [yn] y
restore > quit
[root@beldin]# ls -l passwd
-rw-r--r--   1 root     root          787 Jan 25 15:00 passwd
```

Alternative

Rather than backup to a normal file on the hard-drive you could choose to backup files directly to a floppy drive (i.e. use `/dev/fd0` rather than `/tmp/backup`). One problem with this alternative is that you are limited to 1.44Mb per media. This used to be a problem because the Linux version of dump did not support multiple volumes. It appears it know does.

Exercises

11.1. Do a level 0 dump of a portion of your home directory onto a file somewhere on your hard drive. Examine the file `/etc/dumpdates`. How has it changed?

11.2. Use `restore` to retrieve some individual files from the backup and also to retrieve the entire backup.

11.3. Perform a dump onto floppies which requires more than 1.44Mb of space (the idea here is to play around with multiple volume backups). You can do this by creating a directory and placing files into it until you have more than 1.44Mb of data in it (use the du command to find out how much space is being consumed). After you've backed up onto floppies try retrieving some files. What problems do you face?

The `tar` command

`tar` is a general purpose command used for archiving files. It takes multiple files and directories and combines them into one large file. By default the resulting file is written to a default device (usually a tape drive). However the resulting file can be placed onto a disk drive.

```
tar -function[modifier] device [files]
```

The purpose and values for `function` and `modifier` are shown in Tables 11.5 through 11.7.

When using `tar`, each individual file stored in the final archive is preceded by a header that contains approximately 512 bytes of information. Also the end of the file is always padded so that it occurs on an even block boundary. For this reason, every file added into the tape archive has on average an extra `.75Kb` of padding per file.

Arguments	Purpose
`function`	A single letter specifying what should be done, values listed in Table 11.6
`modifier`	Letters that modify the action of the specified function, values listed in Table 11.7
`files`	The names of the files and directories to be restored or archived. If it is a directory then EVERYTHING in that directory is restored or archived

Table 11.5.
Arguments to tar.

Function	Purpose
c	create a new tape, do not write after last file
r	replace, the named files are written onto the end of the tape
t	table, information about specified files is listed, similar in output to the command ls -l, if no files specified all files listed
u *	update, named files are added to the tape if they are not already there or they have been modified since being previously written
x	extract, named files restored from the tape, if the named file matches a directory all the contents are extracted recursively

* ***the u function can be very slow***

Table 11.6.
Values of the function argument for tar.

Modifier	Purpose
v	verbose, `tar` reports what it is doing and to what
w	`tar` prints the action to be taken, the name of the file and waits for user confirmation
f	file, causes the device parameter to be treated as a file
m	modify, tells tar not to restore the modification times as they were archived but instead to use the time of extraction
o	ownership, use the UID and GID of the user running `tar` not those stored on the tape

Table 11.7.
Values of the modifier argument for tar.

If the `f` modifier is used it must be the last modifier used. Also `tar` is an example of a UNIX command where the `-` character is not required to specify modifiers.

For example:

```
tar -xvf temp.tar              tar xvf temp.tar
```

extracts all the contents of the `tar` file `temp.tar`

```
tar -xf temp.tar hello.dat
```

extracts the file `hello.dat` from the `tar` file `temp.tar`

```
tar -cv /dev/rmt0 /home
```

archives all the contents of the `/home` directory onto tape, overwriting whatever is there

Exercises

11.4. Create a file called `temp.dat` under a directory `tmp` that is within your home directory. Use `tar` to create an archive containing the contents of your home directory.

11.5. Delete the `$HOME/tmp/temp.dat` created in the previous question. Extract the copy of the file that is stored in the tape archive (the term tape archive is used to refer to a file created by `tar`) created in the previous question.

The dd command

The man page for `dd` lists its purpose as being "`copy and convert data`". Basically `dd` takes input from one source and sends it to a different destination. The source and destination can be device files for disk and tape drives, or normal files.

The basic format of dd is

```
dd [option = value ....]
```

Table 11.8. lists some of the different options available.

Option	Purpose
`if=`*name*	input file name (default is standard input)
`of=`*name*	output file name (default is standard output)
`ibs=`*num*	the input block size in *num* bytes (default is 512)
`obs=`*num*	the output block size in *num* bytes (default is 512)
`bs=`*num*	set both input and output block size
`skip=`*num*	skip *num* input records before starting to copy
`files=`*num*	copy *num* files before stopping (used when input is from magnetic tape)
`conv=ascii`	convert EBCDIC to ASCII
`conv=ebcdic`	convert ASCII to EBCDIC
`conv=lcase`	make all letters lowercase
`conv=ucase`	make all letters uppercase
`conv=swab`	swap every pair of bytes

Table 11.8.
Options for dd.

For example:

```
dd if=/dev/hda1 of=/dev/rmt4
```

with all the default settings copy the contents of `hda1` (the first partition on the first disk) to the tape drive for the system

Exercises

11.6. Use `dd` to copy the contents of a floppy disk to a single file to be stored under your home directory. Then copy it to another disk.

The `mt` command

The usual media used in backups is magnetic tape. Magnetic tape is a sequential media. That means that to access a particular file you must pass over all the tape containing files that come before the file you want. The `mt` command is used to send commands to a magnetic tape drive that control the location of the read/write head of the drive.

```
mt [-f tapename] command [count]
```

Arguments	Purpose
tapename	raw device name of the tape device
command	one of the commands specified in table 11.10. Not all commands are recognised by all tape drives.
count	number of times to carry out command

Table 11.9. Parameters for the mt Command.

Commands	Action
`fsf`	move forward the number of files specified by the *count* argument
`asf`	move forward to file number *count*
`rewind`	rewind the tape
`retension`	wind the tape out to the end and then rewind
`erase`	erase the entire tape
`offline`	eject the tape

Table 11.10. Commands Possible using the mt Command.

For example:

```
mt -f /dev/nrst0 asf 3
```

moves to the third file on the tape

```
mt -f /dev/nrst0 rewind
mt -f /dev/nrst0 fsf 3
```

same as the first command

The `mt` command can be used to put multiple `dump/tar` archive files onto the one tape. Each time `dump/tar` is used, one file is written to the tape. The `mt` command can be used to move the read/write head of the tape drive to the end of that file, at which time `dump/tar` can be used to add another file.

For example:

```
mt -f /dev/rmt/4 rewind
```

rewinds the tape drive to the start of the tape

```
tar -cvf /dev/rmt/4 /home/jonesd
```

backs up my home directory, after this command the tape will be automatically rewound

```
mt -f /dev/rmt/4 asf 1
```

moves the read/write head forward to the end of the first file

```
tar -cvf /dev/rmt/4a /home/thorleym
```

backs up the home directory of `thorleym` onto the end of the tape drive

There are now two `tar` files on the tape, the first containing all the files and directories from the directory `/home/jonesd` and the second containing all the files and directories from the directory `/home/thorleym`.

Compression programs

Compression programs are sometimes used in conjunction with transport programs to reduce the size of backups. This is not always a good idea. Adding compression to a backup adds extra complexity to the backup and as such increases the chances of something going wrong.

`compress`

`compress` is the standard UNIX compression program and is found on every UNIX machine (well, I don't know of one that doesn't have it). The basic format of the `compress` command is

```
compress filename
```

The file with the name *`filename`* will be replaced with a file with the same name but with an extension of `.Z` added, and that is smaller than the original (it has been compressed).

A compressed file is uncompressed using the `uncompress` command or the `-d` switch of `compress`.

```
uncompress filename   or   compress -d filename
```

For example:

```
bash$ ls -l ext349*
-rw-r----- 1 jonesd      17340 Jul 16 14:28 ext349
bash$ compress ext349
bash$ ls -l ext349*
-rw-r----- 1 jonesd       5572 Jul 16 14:28 ext349.Z
bash$ uncompress ext349
bash$ ls -l ext349*
-rw-r----- 1 jonesd      17340 Jul 16 14:28 ext349
```

gzip

`gzip` is a new addition to the UNIX compression family. It works in basically the same way as `compress` but uses a different (and better) compression algorithm. It uses an extension of `.z` and the program to uncompress a `gzip` archive is `gunzip`.

For example:

```
bash$ gzip ext349
bash$ ls -l ext349*
-rw-r----- 1 jonesd    4029 Jul 16 14:28 ext349.z
bash$ gunzip ext349
```

Exercises

11.7.	Modify your solution to exercise 11.5 so that instead of writing the contents of your floppy straight to a file on your hard disk it first compresses the file using either `compress` or `gzip` and then saves to a file.

Conclusions

In this chapter you have

- been introduced to the components of a backup strategy scheduler, transport, and media
- been shown some of the UNIX commands that can be used as the transport in a backup strategy
- examined some of the characteristics of a good backup strategy and some of the factors that affect a backup strategy

Review questions

11.1.

Design a backup strategy for your system. List the components of your backup strategy and explain how these components affect your backup strategy.

11.2.

Explain the terms media, scheduler and transport.

11.3.

Outline the difference between file by file and image transport programs.

11.4.

ACME Backup Systems has just produced a wonderful backup system which has caught the eye of your manager. He has decided that it is the product you should be using to backup the Linux servers within your organization. ACME's "Backup-true" product is an image transport developed for the Windows NT file system. Why can't you use it for to backup your Linux systems?

Chapter 12

Startup and Shutdown

Introduction

Being a multi-tasking, multi-user operating system means that UNIX is a great deal more complex than an operating system like MS-DOS. Before the UNIX operating system can perform correctly, there are a number of steps that must be followed, and procedures executed. The failure of any one of these can mean that the system will not start, or if it does it will not work correctly. It is important for the Systems Administrator to be aware of what happens during system startup so that any problems that occur can be remedied.

It is also important for the Systems Administrator to understand what the correct mechanism is to shut a UNIX machine down. A UNIX machine should (almost) never be just turned off. There are a number of steps to carry out to ensure that the operating system and many of its support functions remain in a consistent state.

By the end of this chapter you should be familiar with the startup and shutdown procedures for a UNIX machine and all the related concepts.

Other Resources

There is a lot of available information about the startup process of a Linux machine and also how you recover from errors in the startup process. These include

- HOW-TOs
  BootPrompt HOW-TO, Boot disk HOW-TO, UPS HOW-TO, LILO Mini HOW-TO, Win95 + WinNT + Linux multiboot using LILO mini-HOWTO
- Rescue disk sets
- The Linux Systems Administrator's Guide (part of the LDP and on the 85321 CD-ROM) chapter (chapter 6) on boots and shutdowns and also Chapter 7 on init.

A booting overview

The process by which a computer is turned on and the UNIX operating system starts functioning – booting - consists of the following steps

- finding the kernel,
  The first step is to find the kernel of the operating system. How this is achieved is usually particular to the type of hardware used by the computer.

- starting the kernel,
  In this step the kernel starts operation and in particular goes looking for all the hardware devices that are connected to the machine.
- starting the processes.
  All the work performed by a UNIX computer is done by processes. In this stage, most of the system processes and daemons are started. This step also includes a number of steps which configure various services necessary for the system to work.

Finding the Kernel

For a UNIX computer to be functional it must have a kernel. The kernel provides a number of essential services which are required by the rest of the system in order for it to be functional. This means that the first step in the booting process of a UNIX computer is finding out where the kernel is. Once found, it can be started, but that's the next section.

ROM

Most machines have a section of read only memory (ROM) that contains a program the machine executes when the power first comes on. What is programmed into ROM will depend on the hardware platform.

For example, on an IBM PC, the ROM program typically does some hardware probing and then looks in a number of predefined locations (the first floppy drive and the primary hard drive partition) for a bootstrap program.

On hardware designed specifically for the UNIX operating system (machines from DEC, SUN etc), the ROM program will be a little more complex. Many will present some form of prompt. Generally this prompt will accept a number of commands that allow the Systems Administrator to specify

- where to boot the machine from, and
  Sometimes the standard root partition will be corrupt and the system will have to be booted from another device. Examples include another hard drive, a CD-ROM, floppy disk or even a tape drive.
- whether to come up in single user or multi-user mode.

As a bare minimum, the ROM program must be smart enough to work out where the bootstrap program is stored and how to start executing it.

The ROM program generally doesn't know enough to know where the kernel is or what to do with it.

The bootstrap program

At some stage the ROM program will execute the code stored in the boot block of a device (typically a hard disk drive). The code stored in the boot block is referred to as a bootstrap program. Typically the boot block isn't big enough to hold the kernel of an operating system so this intermediate stage is necessary.

The bootstrap program is responsible for locating and loading (starting) the kernel of the UNIX operating system into memory. The kernel of a UNIX

operating system is usually stored in the root directory of the root file system under some system-defined filename. Newer versions of Linux, including RedHat 5.0 and above, put the kernel into a directory called `/boot`. /boot is often on a separate partition. In fact the default installation of RedHat Linux will create /boot as a separate partition.

The most common bootstrap program in the Linux world is a program called LILO.

Reading

LILO is such an important program to the Linux operating system that it has its own HOW-TO. The HOW-TO provides a great deal of information about the boot process of a Linux computer.

Booting on a PC

The BIOS on a PC generally looks for a bootstrap program in one of two places (usually in this order)

- the first (`A:`) floppy drive, or
- the first (`C:`) hard drive.

By playing with your BIOS settings you can change this order or even prevent the BIOS from checking one or the other.

The BIOS loads the program that is on the first sector of the chosen drive and loads it into memory. This bootstrap program then takes over. For example, making sure people can't boot your Linux machine of a floppy can prevent them from gaining access to the data on your machine.

On the floppy

On a bootable floppy disk the bootstrap program simply knows to load the first blocks on the floppy that contain the kernel into a specific location in memory.

A normal Linux boot floppy contains no file system. It simply contains the kernel copied into the first sectors of the disk. The first sector on the disk contains the first part of the kernel which knows how to load the remainder of the kernel into RAM.

This means you can't mount the boot floppy onto your Linux machine and read the contents of the disk using ls and other associated commands.

Making a boot disk

The simplest method for creating a floppy disk which will enable you to boot a Linux computer is

- insert a floppy disk into a computer already running Linux
- login as `root`
- change into the `/boot` directory

- copy the current kernel onto the floppy
  `dd if=vmlinuz of=/dev/fd0`
  The name of the kernel, `vmlinuz`, may change from system to system. For example, on some version 5.0 of RedHat Linux it will have been `vmlinux-2.0.31`.

Exercises

12.1.	Using the above steps create a boot floppy for your machine and test it out.

Using a boot loader

Having a boot floppy for your system is a good idea. It can come in handy if you do something to your system which prevents the normal boot procedure from working. One example of this is when you are compiling a new kernel. It is not unheard of for people to create a kernel which will not boot their system. If you don't have an alternative boot method in this situation then you will have some troubles.

However, you can't use this process to boot from a hard-drive. Instead a boot loader or boot strap program, such as LILO, is used. A boot loader generally examines the partition table of the hard-drive, identifies the active partition, and then reads and starts the code in the boot sector for that partition. This is a simplification. In reality the boot loader must identify, somehow, the sectors in which the kernel resides.

Other features a boot loader (under Linux) offers include

- using a key press to bring up a prompt to modify the boot procedure, and
- the passing of parameters to the kernel to modify its operation

Exercises

12.2.	If you have the time, haven't done so already, or know it is destined to failure read the LILO documentation and install LILO onto your system. There are some situations where you SHOULD NOT install LILO. These are outlined in the documentation. Make sure you take notice of these situations.

Starting the kernel

Okay, the boot strap program or the ROM program has found your system's kernel. What happens during the startup process? The kernel will go through the following process

- initialise its internal data structures,
  Things like ready queues, process control blocks and other data structures need to be readied.
- check for the hardware connected to your system,
  It is **important** that you are aware that the kernel will only look for

hardware that it contains code for. If your system has a SCSI disk drive interface your kernel must have the SCSI interface code before it will be able to use it.

- verify the integrity of the root file system and then mount it, and
- create the process 0 (`swapper`) and process 1 (`init`).

The `swapper` process is actually part of the kernel and is not a "real" process. The `init` process is the ultimate parent of all processes that will execute on a UNIX system.

Once the kernel has initialised itself, `init` will perform the remainder of the startup procedure.

Kernel boot messages

When a UNIX kernel is booting, it will display messages on the main console about what it is doing. Under Linux, these messages are also sent to the file `/var/log/dmesg`. The following is a copy of the boot messages on my machine.

Examine the messages that your kernel displays during bootup and compare them with mine.

```
Linux version 2.2.12-20 (root@porky.devel.redhat.com) (gcc version egcs-2.91.66
19990314/Linux (egcs-1.1.2 release)) #1 Mon Sep 27 10:25:54 EDT 1999
Detected 233867806 Hz processor.
Console: colour VGA+ 80x25
Calibrating delay loop... 466.94 BogoMIPS
Memory: 127668k/131072k available (1008k kernel code, 412k reserved, 1640k data, 64k
init)
DENTRY hash table entries: 262144 (order: 9, 2097152 bytes)
Buffer-cache hash table entries: 131072 (order: 7, 524288 bytes)
Page-cache hash table entries: 32768 (order: 5, 131072 bytes)
VFS: Diskquotas version dquot_6.4.0 initialized
CPU: Intel Mobile Pentium MMX stepping 01
Checking 386/387 coupling... OK, FPU using exception 16 error reporting.
Checking 'hlt' instruction... OK.
Intel Pentium with F0 0F bug - workaround enabled.
POSIX conformance testing by UNIFIX
PCI: PCI BIOS revision 2.10 entry at 0xf56ee
PCI: Using configuration type 1
PCI: Probing PCI hardware
PCI: Enabling I/O for device 00:0a
Linux NET4.0 for Linux 2.2
Based upon Swansea University Computer Society NET3.039
NET4: Unix domain sockets 1.0 for Linux NET4.0.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
TCP: Hash tables configured (ehash 131072 bhash 65536)
Initializing RT netlink socket
Starting kswapd v 1.5
Detected PS/2 Mouse Port.
Serial driver version 4.27 with MANY_PORTS MULTIPORT SHARE_IRQ enabled
ttyS00 at 0x03f8 (irq = 4) is a 16550A
pty: 256 Unix98 ptys configured
apm: BIOS version 1.2 Flags 0x03 (Driver version 1.9)
Real Time Clock Driver v1.09
RAM disk driver initialized:  16 RAM disks of 4096K size
PIIX4: IDE controller on PCI bus 00 dev 09
PIIX4: not 100% native mode: will probe irqs later
    ide0: BM-DMA at 0xfcd0-0xfcd7, BIOS settings: hda:pio, hdb:pio
    ide1: BM-DMA at 0xfcd8-0xfcdf, BIOS settings: hdc:pio, hdd:pio
hda: IBM-DCXA-210000, ATA DISK drive
hdc: TOSHIBA CD-ROM XM-1602B, ATAPI CDROM drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
ide1 at 0x170-0x177,0x376 on irq 15
hda: IBM-DCXA-210000, 9590MB w/420kB Cache, CHS=1222/255/63
```

```
hdc: ATAPI 20X CD-ROM drive, 128kB Cache
Uniform CDROM driver Revision: 2.56
Floppy drive(s): fd0 is 1.44M
FDC 0 is a National Semiconductor PC87306
md driver 0.90.0 MAX_MD_DEVS=256, MAX_REAL=12
raid5: measuring checksumming speed
raid5: MMX detected, trying high-speed MMX checksum routines
   pII_mmx   :   340.614 MB/sec
   p5_mmx    :   405.003 MB/sec
   8regs     :   246.888 MB/sec
   32regs    :   184.785 MB/sec
using fastest function: p5_mmx (405.003 MB/sec)
scsi : 0 hosts.
scsi : detected total.
md.c: sizeof(mdp_super_t) = 4096
Partition check:
 hda: hda1 hda2 < hda5 hda6 >
RAMDISK: Compressed image found at block 0
autodetecting RAID arrays
autorun ...
... autorun DONE.
VFS: Mounted root (ext2 filesystem).
autodetecting RAID arrays
autorun ...
... autorun DONE.
VFS: Mounted root (ext2 filesystem) readonly.
change_root: old root has d_count=1
Trying to unmount old root ... okay
Freeing unused kernel memory: 64k freed
Adding Swap: 72252k swap-space (priority -1)
ad1848/cs4248 codec driver Copyright (C) by Hannu Savolainen 1993-1996
[MSS: IRQ Conflict?]
ad1848: Interrupt test failed (IRQ7)
YM3812 and OPL-3 driver Copyright (C) by Hannu Savolainen, Rob Hooft 1993-1996
```

The last few lines of this output demonstrates one of the advantages of checking the kernel boot messages. I've just discovered that the sound configuration for my system is not working as expected. Something I need to investigate and fix.

Starting the processes

So at this stage the kernel has been loaded, it has initialised its data structures and found all the hardware devices. At this stage your system can't do anything. The operating system kernel only supplies services which are used by processes. The question is how are these other processes created and executed.

On a UNIX system the only way in which a process can be created is by an existing process performing a fork operation. A fork creates a brand new process that contains copies of the code and data structures of the original process. In most cases the new process will then perform an `exec` that replaces the old code and data structures with that of a new program.

But who starts the first process?

`init` is the process that is the ultimate ancestor of all user processes on a UNIX system. It always has a Process ID (PID) of 1. `init` is started by the operating system kernel so it is the only process that doesn't have a process as a parent. `init` is responsible for starting all other services provided by the UNIX system. The services it starts are specified by `init`'s configuration file, `/etc/inittab`.

Run levels

`init` is also responsible for placing the computer into one of a number of run levels. The run level a computer is in controls what services are started (or stopped) by `init`. Table 12.2 summarises the different run levels used by RedHat Linux. At any one time, the system must be in one of these run levels.

Run level	Description
`0`	Halt the machine
`1`	Single user mode. All file systems mounted, only small set of kernel processes running. Only `root` can login.
`2`	multi-user mode , without remote file sharing
`3`	multi-user mode with remote file sharing, processes, and daemons
`4`	user definable system state
`5`	used for to start X11 on boot
`6`	shutdown and reboot
`a b c`	ondemand run levels
`s` or `S`	same as single-user mode, only really used by scripts

Table 12.1
Run levels

When a Linux system boots, `init` examines the `/etc/inittab` file for an entry of type `initdefault`. This entry will determine the initial run level of the system.

Under Linux, the `telinit` command is used to change the current run level. `telinit` is actually a soft link to `init`. `telinit` accepts a single character argument from the following

- `0 1 2 3 4 5 6`
  The run level is switched to this level.
- `Q q`
  Tells `init` that there has been a change to `/etc/inittab` (its configuration file) and that it should re-examine it.
- `S s`
  Tells `init` to switch to single user mode.

/etc/inittab

/etc/inittab is the configuration file for init. It is a colon delimited field where # characters can be used to indicate comments. Each line corresponds to a single entry and is broken into four fields

- the identifier
  One or two characters to uniquely identify the entry.
- the run level
  Indicates the run level at which the process should be executed
- the action
  Tells init how to execute the process
- the process
  The full path of the program or shell script to execute.

What happens

When init is first started it determines the current run level (by matching the entry in /etc/inittab with the action initdefault) and then proceeds to execute all of the commands of entries that match the run level.

The following is an example /etc/inittab taken from a RedHat machine with some comments added.

Specify the default run level

```
id:3:initdefault:

# System initialisation.
si::sysinit:/etc/rc.d/rc.sysinit
```

when first entering various runlevels run the related startup scripts before going any further

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Things to run in every runlevel.
ud::once:/sbin/update
```

call the shutdown command to reboot the system when the use does the three fingered salute

```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

A powerfail signal will arrive if you have a uninterruptable power supply (UPS) if this happens shut the machine down safely

```
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
```

Start the login process for the virtual consoles

```
1:12345:respawn:/sbin/mingetty tty1
```

```
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

If the machine goes into runlevel 5, start X

```
x:5:respawn:/usr/bin/X11/xdm -nodaemon
```

The identifier

The identifier, the first field, is a unique two character identifier. For `inittab` entries that correspond to terminals the identifier will be the suffix for the terminals device file.

For each terminal on the system a `mingetty` process must be started by the `init` process. Each terminal will generally have a device file with a name like `/dev/tty??`, where the `??` will be replaced by a suffix. It is this suffix that must be the identifier in the `/etc/inittab` file.

Run levels

The run levels describe at which run levels the specified action will be performed. The run level field of `/etc/inittab` can contain multiple entries, e.g. `123`, which means the action will be performed at each of those run levels.

Actions

The action's field describes how the process will be executed. There are a number of pre-defined actions that must be used. Table 10.2 lists and explains them.

Action	Purpose
respawn	restart the process if it finishes
wait	init will start the process once and wait until it has finished before going on to the next entry
once	start the process once, when the runlevel is entered
boot	perform the process during system boot (will ignore the runlevel field)
bootwait	a combination of boot and wait
off	do nothing
initdefault	specify the default run level
sysinit	execute process during boot and before any boot or bootwait entries
powerwait	executed when init receives the SIGPWR signal which indicates a problem with the power, init will wait until the process is completed
ondemand	execute whenever the ondemand runlevels are called (a b c). When these runlevels are called there is NO change in runlevel.
powerfail	same as powerwait but don't wait (refer to the man page for the action powerokwait)
ctrlaltdel	executed when init receives SIGINT signal (usually when someone does CTRL-ALT-DEL

Table 12.2
inittab actions

The process

The process is simply the name of the command or shell script that should be executed by init.

Daemons and Configuration Files

init is an example of a daemon. It will only read its configuration file, /etc/inittab, when it starts execution. Any changes you make to /etc/inittab will not influence the execution of init until the next time it starts, i.e. the next time your computer boots.

There are ways in which you can tell a daemon to re-read its configuration files. One generic method, which works most of the time, is to send the daemon the HUP signal. For most daemons the first step in doing this is to find out what the process id (PID) is of the daemon. This isn't a problem for init. Why?

It's not a problem for init because init always has a PID of 1.

The more accepted method for telling `init` to re-read its configuration file is to use the `telinit` command. `telinit q` will tell `init` to re-read its configuration file.

Exercises

12.3. Add an entry to the `/etc/inittab` file so that it displays a message `HELLO` onto your current terminal (HINT: you can find out your current terminal using the `tty` command).

12.4. Modify the `inittab` entry from the previous question so that the message is displayed again and again and....

12.5. Take your system into single user mode.

12.6. Take your system into runlevel 5. What happens? (***only do this if you have X Windows configured for your system***). Change your system so that it enters this run level when it boots. Reboot your system and see what happens.

12.7. The `wall` command is used to display a message onto the terminals of all users. Modify the `/etc/inittab` file so that whenever someone does the three finger salute (`CTRL-ALT-DEL`) it displays a message on the consoles of all users and doesn't log out.

12.8. Examine your `inittab` file for an entry with the identifier `c1`. This is the entry for the first console, the screen you are on when you first start your system.
Change the entry for `c1` so that the action field contains `once` instead of `respawn`. Force `init` to re-read the `inittab` file and then log in and log out on that console.
What happens?

System Configuration

There are a number of tasks which must be completed once during system startup which must be completed once. These tasks are usually related to configuring your system so that it will operate. Most of these tasks are performed by the `/etc/rc.d/rc.sysinit` script.

It is this script which performs the following operations

- sets up a search path that will be used by the other scripts
- obtains network configuration data
- activates the swap partitions of your system
- sets the hostname of your system
  Every UNIX computer has a hostname. You can use the UNIX command `hostname` to set and also display your machine's hostname.
- sets the machines NIS domain (if you are using one)
- performs a check on the file systems of your system

- turns on disk quotas (if being used)
- sets up plug'n'play support
- deletes old lock and tmp files
- sets the system clock
- loads any kernel modules.

Terminal logins

For a user to login there must be a `getty` process (RedHat Linux uses a program called `mingetty`, slightly different name but same task) running for the terminal they wish to use. It is one of `init`'s responsibilities to start the `getty` processes for all terminals that are physically connected to the main machine, and you will find entries in the `/etc/inittab` file for this.

Please note this does not include connections over a network. They are handled with a different method. This method is used for the virtual consoles on your Linux machine and any other dumb terminals you might have connected via serial cables. You should be able see the entries for the virtual consoles in the example `/etc/inittab` file from above.

Exercises

12.9.	When you are in single user mode there is only one way to login to a Linux machine, from the first virtual console. How is this done?

Startup scripts

Most of the services which `init` starts are started when `init` executes the system start scripts. The system startup scripts are shell scripts written using the Bourne shell (this is one of the reasons you need to know the bourne shell syntax). You can see where these scripts are executed by looking at the `inittab` file.

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

These scripts start a number of services and also perform a number of configuration checks including

- checking the integrity of the machine's file systems using `fsck`,
- mounting the file systems,
- designating paging and swap areas,
- checking disk quotas,

- clearing out temporary files in `/tmp` and other locations,
- startin up system daemons for printing, mail, accounting, system logging, networking, `cron` and `syslog`.

In the UNIX world there are two styles for startup files: BSD and System V. RedHat Linux uses the System V style and the following section concentrates on this format. Table 12.3 summarises the files and directories which are associated with the RedHat startup scripts. All the files and directories in Table 12.3 are stored in the `/etc/rc.d` directory.

Filename	Purpose
`rc0.d rc1.d rc2.d rc3.d rc4.d rc5.d rc6.d`	directories which contain links to scripts which are executed when a particular runlevel is entered
`rc`	A shell script which is passed the run level. It then executes the scripts in the appropriate directory.
`init.d`	Contains the actual scripts which are executed. These scripts take either `start` or `stop` as a parameter
`rc.sysinit`	run once at boot time to perform specific system initialisation steps
`rc.local`	the last script run, used to do any tasks specific to your local setup that isn't done in the normal SysV setup
`rc.serial`	not always present, used to perform special configuration on any serial ports

Table 12.3
Linux startup scripts

The Linux Process

When `init` first enters a run level it will execute the script `/etc/rc.d/rc` (as shown in the example `/etc/inittab` above). This script then proceeds to

- determine the current and previous run levels
- kill any services which must be killed
- start all the services for the new run level.

The `/etc/rc.d/rc` script knows how to kill and start the services for a particular run level because of the filenames in the directory for each runlevel. The following are the filenames from the `/etc/rc.d/rc3.d` directory on my system.

```
[david@beldin rc.d]$ ls rc3.d
K20rstatd   S05kudzu             S20random  S40crond   S75keytable
S99linuxconf
K20rusersd  S10network     S25netfs   S45pcmcia  S80sendmail  S99local
K20rwhod    S11portmap     S30syslog  S50inet    S85gpm
K55routed   S16apmd        S40atd     S60lpd     S90xfs
```

You will notice that all the filenames in this, and all the other `rcX.d` directories, use the same format.

`[SK]`*`numberService`*

Where *`number`* is some integer and *`Service`* is the name of a service.

All the files with names starting with `S` are used to start a service. Those starting with `K` are used to kill a service. From the `rc3.d` directory above you can see scripts which start services for the Internet (`S50inet`), PCMCIA cards (`S45pcmcia`), a and others.

The numbers in the filenames are used to indicate the order in which these services should be started and killed. You'll notice that the script to start the Internet services comes before the script to start the Web server; obviously the Web server depends on the Internet services.

`/etc/rc.d/init.d`

If we look closer we can see that the files in the `rcX.d` directories aren't really files.

```
[david@beldin rc.d]$ ls -l rc3.d/S50inet
lrwxrwxrwx   1 root   root  14 Dec 19 23:57 rc3.d/S50inet -> ../init.d/inet
```

The files in the `rcX.d` directories are actually soft links to scripts in the `/etc/rc.d/init.d` directory. It is these scripts which perform all the work.

Starting and stopping

The scripts in the `/etc/rc.d/init.d` directory are not only useful during the system startup process, they can also be useful when you are performing maintenance on your system. You can use these scripts to start and stop services while you are working on them.

For example, lets assume you are changing the configuration of your Web server. Once you've finished editing the configuration files you will need to restart the Web server for it to see the changes. One way you could do this would be to follow this example

```
[root@beldin rc.d]# /etc/rc.d/init.d/httpd stop
Shutting down http:
[root@beldin rc.d]# /etc/rc.d/init.d/httpd start
Starting httpd: httpd
```

This example also shows you how the scripts are used to start or stop a service. If you examine the code for `/etc/rc.d/rc` (remember this is the script which runs all the scripts in `/etc/rc.d/rcX.d`) you will see two lines. One with `$i start` and the other with `$i stop`. These are the actual lines which execute the scripts.

Lock files

All of the scripts which start services during system startup create lock files. These lock files, if they exist, indicate that a particular service is operating. Their main use is to prevent startup files starting a service which is already running.

When you stop a service one of the things which has to occur is that the lock file must be deleted.

Exercises

12.10. What would happen if you tried to stop a service when you were logged in as a normal user (i.e. not root)? Try it.

Why won't it boot?

There will be times when you have to reboot your machine in a nasty manner. One rule of thumb used by Systems Administration to solve some problems is "When in doubt, turn the power off, count to ten slowly, and turn the power back on". There will be times when the system won't come back to you, **DON'T PANIC!**

Possible reasons why the system won't reboot include

- hardware problems,
  Caused by both hardware failure and problems caused by human error (e.g. the power cord isn't plugged in, the drive cable is the wrong way around)
- defective boot floppies, drives or tapes,
- damaged file systems,
- improperly configured kernels,
  A kernel configured to use SCSI drives won't boot on a system that uses an IDE drive controller.
- errors in the rc scripts or the `/etc/inittab` file.

Solutions

The following is a Systems Administration maxim

> *Always keep a separate working method for booting the machine into at least single user mode.*

This method might be a boot floppy, CD-ROM or tape. The format doesn't matter. What does matter that at anytime you can bring the system up in at least single user mode so you can perform some repairs.

A separate mechanism to bring the system up single user mode will enable you to solve most problems involved with damaged file systems, improperly configured kernels and errors in the rc scripts.

Boot and root disks

The concept of boot and root disk are important to understanding how the booting process works and also in creating an alternative boot method for your system. The definitions used are

- boot disk
  This is the disk which contains the kernel of your system.

- root disk
  The root disk contains the root file system with all the necessary programs and files required for `init` to start and setup a minimum of services. This includes such things as `init`, `/etc/inittab` and associated files, `/etc/passwd` and other information required to allow people to login plus a whole lot more.

To have a complete alternative boot method you must have both alternative boot and root disks. The alternative boot disk is useful if you have problems with your kernel. The alternative root disk is required when you have problems such as a wrongly configured `inittab` or a missing `/etc/passwd` file.

It is possible for a single disk to provide both boot and root disk services.

Making a boot and root disk

It is important that you have alternative boot and root disks for your system. There are (at least) two methods you can use to obtain them

- use the installation disks which come with your distribution of Linux,
  In order to install Linux you basically have to have a functioning Linux computer. Therefore the installation disk(s) that you used to install Linux provide an alternative boot and root disk.
- use a rescue disk (set).
  A number of people have created rescue disks. These are boot and root disk sets which have been configured to provide you with the tools you will need to rescue your system from problems.

The resource materials section for week 7 on the 85321 Web site/CD-ROM contains pointers to two rescue disk sets.

Exercises

12.11.	Create a boot and root disk set for your system using the resources on the 85321 Web site/CD-ROM.

Using boot and root

What do you think would happen if you did the following?

```
rm /etc/inittab
```

The next time you booted your system you would see something like this on the screen.

```
INIT: version 2.71 booting
INIT: No inittab file found

Enter runlevel: 1
INIT: Entering runlevel: 1
INIT: no more processes left in this runlevel
```

What's happening here is that `init` can't find the `inittab` file and so it can't do anything. To solve this you need to boot the system and replace the missing

`inittab` file. This is where the alternative root and boot disk(s) come in handy.

To solve this problem you would do the following

- boot the system with the alternative boot/root disk set
- login as `root`
- perform the following

```
/> mount –t ext2 /dev/hda2 /mnt
mount: mount point /mnt does not exist
/> mkdir /mnt
/> mount –t ext2 /dev/hda1 /mnt
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended
/> cp /etc/inittab /mnt/etc/inittab
/> umount /mnt
```

A description of the above goes like this

- Try to mount the usual root file system, the one with the missing `inittab` file. But it doesn't work.
- Create the missing `/mnt` directory.
- Now mount the usual root file system.
- Copy the `inittab` file from the alternative root disk onto the usual root disk. Normally you would have a backup tape which contains a copy of the old `inittab` file.
- Unmount the usual root file system and reboot the system.

The aim of this example is to show you how you can use alternative root and boot disks to solve problems which may prevent your system from booting.

Exercises

12.12.	Removing the `/etc/inittab` file from your Linux system will not only cause problems when you reboot the machine. It also causes problems when you try to shut the machine down. What problems? Why?
12.13.	What happens if you forget the root password? Without it you can't perform any management tasks at all. How would you fix this problem?
12.14.	Boot your system in the normal manner and comment out all the entries in your `/etc/inittab` file that contain the word `mingetty`. What do you think is going to happen? Reboot your system. Now fix the problem using the installation floppy disks.

Solutions to hardware problems

Some guidelines to solving hardware problems

- check the power supply and its connections,
  Don't laugh, there are many cases I know of in which the whole problem was caused by the equipment not being plugged in properly or not at all.
- check the cables and plugs on the devices,
- check any fault lights on the hardware,
- power cycle the equipment (power off, power on),
  There is an old Systems Administration maxim. If something doesn't work turn it off, count to 10 very slowly and turn it back on again (usually with the fingers crossed). Not only can it solve problems but it is also a good way of relaxing. Of course this is a last resort and in some cases may not be available. For example, if you are in charge of a machine which is required to have 24x7 availability.
- try rebooting the system without selected pieces of hardware,
  It may be only one faulty device that is causing the problem. Try isolating the problem device.
- use any diagnostic programs that are available, or as a last resort
- call a technician or a vendor.

Damaged file systems

Previous chapters examined file systems and backups. Fixing a damaged file system involved first trying to use the fsck command and if that fails using backups.

Improperly configured kernels

The kernel contains most of the code that allows the software to talk to your hardware. If the code it contains is wrong then your software won't be able to talk to your hardware. In a later chapter on the kernel we'll explain in more detail why you might want to change the kernel and why it might not work.

Suffice to say you must **always** maintain a working kernel that you can boot your system with.

Shutting down

You should not just simply turn a UNIX computer off or reboot it. Doing so will usually cause some sort of damage to the system especially to the file system. Most of the time the operating system may be able to recover from such a situation (but NOT always).

There are a number of tasks that have to be performed for a UNIX system to be shutdown cleanly

- tell the users the system is going down,
  Telling them 5 seconds before pulling the plug is not a good way of promoting good feeling amongst your users. Wherever possible the users should know at least a couple of days in advance that the system is going down (there is always one user who never knows about it and complains).
- signal the currently executing processes that it is time for them to die,
  UNIX is a multi-tasking operating system. Just because there is no-one logged in this does not mean that there is nothing going on. You must signal all the current running processes that it is time to die gracefully.
- place the system into single user mode, and
- perform sync to flush the file systems buffers so that the physical state of the file system matches the logical state.

Most UNIX systems provide commands that perform these steps for you.

As computers become more important to the operation of a business systems must have 24x7 availability. Imagine how much money Amazon.com or eBay lose if and when their computers are unavailable. In these situations shutting down a computer usually involves ensuring that there is another computer already running which will take over operations.

Reasons Shutting down

In general, you should try to limit the number of times you turn a computer on or off as doing so involves some wear and tear. It is often better to simply leave the computer on 24 hours a day. In the case of a UNIX system being used for a mission critical application by some business it may have to be up 24 hours a day.

Some of the reasons why you may wish to shut a UNIX system down include

- general housekeeping,
  Every time you reboot a UNIX computer it will perform some important housekeeping tasks, including deleting files from the temporary directories and performing checks on the machines file systems. Rebooting will also get rid of any zombie processes.
- general failures, and
  Occasionally problems will arise for which there is only one resort, shutdown. These problems can include hanging logins, unsuccessful mount requests, dazed devices, runaway processes filling up disk space or CPU time and preventing any useful work being done.
- system maintenance and additions.
  There are some operations that only work if the system is rebooted or if the system is in single user mode, for example adding a new device.

Being nice to the users

Knowing of the existence of the appropriate command is the first step in bringing your UNIX computer down. The other step is outlined in the heading for this section. The following command is an example of what not to do.

```
shutdown -h -1 now
```

Under Linux this results in a message somewhat like this appearing on every user's terminal

```
THE SYSTEM IS BEING SHUT DOWN NOW ! ! !
Log off now or risk your files being damaged.
```

and the user will almost immediately be logged out.

This is not a method inclined to win friends and influence people. The following is a list of guidelines of how and when to perform system shutdowns

- shutdowns should be scheduled,
  If users know the system is coming down at specified times they can organise their computer time around those times.
- perform a regular shutdown once a week, and
  A guideline, so that the housekeeping tasks discussed above can be performed. If it's regular the users get to know when the system will be going down.
- use `/etc/motd`.
  `/etc/motd` is a text file that contains the message the users see when they first log onto a system. You can use it to inform users of the next scheduled shutdown.
- /etc/motd is rarely used these days. It's really only visible when you logon to the command line of a UNIX machine. Not to many people do that these days. Alternatives are available including increased use of staff mailing lists.

Commands to shutdown

There are a number of different methods for shutting down and rebooting a system including

- the `shutdown` command
  The most used method for shutting the system down. The command can display messages at preset intervals warning the users that the system is coming down.
  Most Linux computers are configured so that the three-fingered salute (CTRL-ALT-DEL) will automatically cause the shutdown command to be executed. Refer back to your /etc/inittab file and see if you can see the entry for it.
- the `halt` command
  Logs the shutdown, kills the system processes, executes `sync` and halts the processor.
- the `reboot` command
  Similar to halt but causes the machine to reboot rather than halting.
- sending `init` a `TERM` signal,
  `init` will usually interpret a `TERM` signal (signal number 15) as a command to go into single user mode. It will kill of user processes and daemons. The

command is `kill -15 1` (`init` is always process number 1). It may not work or be safe on all machines.

- the `fasthalt` or `fastboot` commands
  These commands create a file `/fastboot` before calling `halt` or `reboot`. When the system reboots and the startup scripts find a file `/fastboot` they will not perform a `fsck` on the file systems.

The most used method will normally be the `shutdown` command. It provides users with warnings and is the safest method to use.

`shutdown`

The format of the command is

```
shutdown [ -h | -r ] [ -fqs ] [ now | hh:ss | +mins ]
```

The parameters are

- `-h`
  Halt the system and don't reboot.
- `-r`
  Reboot the system
- `-f`
  Do a fast boot.
- `-q`
  Use a default broadcast message.
- `-s`
  Reboot into single user mode by creating a `/etc/singleboot` file.

`The time at which a shutdown should occur are specified by the now hh:ss +mins` options.

- `now`
  Shut down immediately.
- `hh:ss`
  Shut down at time `hh:ss`.
- `+mins`
  Shut down `mins` minutes in the future.

The default wait time before shutting down is two minutes.

What happens

The procedure for shutdown is as follows

- five minutes before shutdown or straight away if shutdown is in less than five minutes
  The file `/etc/nologin` is created. This prevents any users (except `root`) from logging in. A message is also broadcast to all logged in users notifying them of the imminent shutdown.

- at shutdown time.
  All users are notified. `init` is told not to spawn any more `getty` processes. Shutdown time is written into the file `/var/log/wtmp`. All other processes are killed. A `sync` is performed. All file systems are unmounted. Another `sync` is performed and the system is rebooted.

The other commands

The other related commands including `reboot, fastboot, halt, fasthalt` all use a similar format to the `shutdown` command. Refer to the man pages for more information.

Conclusions

Booting and shutting down a UNIX computer is significantly more complex than performing the same tasks with a MS-DOS computer. A UNIX computer should never just be shut off.

The UNIX boot process can be summarised into a number of steps

- the hardware ROM or BIOS performs a number of tasks including loading the bootstrap program,
- the bootstrap program loads the kernel,
- the kernel starts operation, configures the system and runs the `init` process
- `init` consults the `/etc/inittab` file and performs a number of necessary actions.

One of the responsibilities of the `init` process is to execute the startup scripts that, under Linux, reside in the `/etc/rc.d` directory.

It is important that you have at least one other alternative method for booting your UNIX computer.

There are a number of methods for shutting down a UNIX computer. The most used is the `shutdown` command.

Review Questions

12.1

What would happen if the file `/etc/inittab` did not exist? Find out.

12.2

How would you fix the following problems?

- The kernel for your Linux computer has been accidentally deleted.
- The `/etc/fstab` file for your system has been moved to `/usr/local/etc/fstab`.

12.3

Explain each of the following `inittab` entries

- `s1:45:respawn:/sbin/agetty 19200 ttyS0 vt100`

- `id:5:initdefault:`
- `si:S:sysinit:/etc/rc.d/rc.S`

12.4

Your boss has decided that you should prevent people from being able to boot your Linux server via a floppy by modifying the BIOS configuration. Why does this increase the security of the server somewhat? What problems might this approach have?

Chapter 13

Kernel

The bit of the nut that you eat?

Well, not exactly. The kernel is the core of the operating system; it is the program that controls the basic services that are utilised by user programs; it is this suite of basic services in the form of system calls that make an operating system "UNIX".

The kernel is also responsible for:

- CPU resource scheduling (with the associated duties of process management)
- Memory management (including the important implementation of protection)
- Device control (including providing the device-file/device-driver interface)
- Security (at a device, process and user level)
- Accounting services (including CPU usage and disk quotas)
- Inter Process Communication (shared memory, semaphores and message passing)

The Linux Kernel FAQ sums it up nicely with:

```
The Unix kernel acts as a mediator for your programs. First, it does
the memory management for all of the running programs (processes),
and makes sure that they all get a fair (or unfair, if you please)
share of the processor's cycles. In addition, it provides a nice,
fairly portable interface for programs to talk to your hardware.

Obviously, there is more to the kernel's operation than this, but the
basic functions above are the most important to know.
```

Other Resources

Other resources which discuss kernel related matters include

- HOW-TOs
  Kernel HOWTO, Kerneld mini-HOWTO, LILO mini-HOWTO, Modules mini HOW-TO
- The Linux Kernel
  A book, available from the LDP (on the 85321 CD-ROM), which describes the principles and mechanisms used by the Linux Kernel (version 2.0.33).
- Linux Kernel Module Programming Guide
  A book, available from the LDP, which describes how to write kernel modules.

- Linux Device Drivers
  A book from O'Relly describing how to write device drivers for Linux.
  http://www.ora.com/catalog/linuxdrive/
- LAME
  A book from the LDP which includes sections on Linux Kernel Upgrades, Upgrading a Red Hat Stock Kernel, Building a Custom Kernel, and Moving to the Linux 2.2.x Kernels.
- The RedHat 6.1 Reference Guide
  Includes a number of sections describing the process for configuring and compiling kernels.
- The Linux Kernel Archives
  http://www.kernel.org/ The primary site for the source of the Linux kernel.
- The International Kernel Patch
  http://www.kerneli.org/ Where the Linux kernel, with fully-fledged cryptographic support, is distributed (sites in the US can't legally distribute it).
- Kernel Notes
  http://www.kernelnotes.org/ A collection of very good links to everything Linux and kernel related.

Why?

Why study the kernel? Isn't that an operating-system-type-thing? What does a Systems Administrator have to do with the internal mechanics of the OS?

Lots.

UNIX is usually provided with the source for the kernel (there are exceptions to this in the commercial UNIX world). The reason is that this allows Systems Administrators to directly customise the kernel for their particular system. A Systems Administrator might do this because:

- They have modified the system hardware (adding devices, memory, processors etc.).
- They wish to optimise the memory usage (called reducing the kernel footprint).
- The speed and performance of the system may need improvement (eg. modify the quantum per task to suit CPU intensive vs IO intensive systems). This process (along with optimising memory) is called tweaking.
- Improvements to the kernel can be provided in the form of source code which then allows the Systems Administrator to easily upgrade the system with a kernel recompile.

> Recompiling the kernel is the process whereby the kernel is reconfigured, the source code is regenerated/recompiled and a linked object is produced. Throughout this chapter the concept of recompiling the kernel will mean both the kernel source code compilation and linkage.

How?

In this chapter, we will be going through the step-by-step process of compiling a kernel, a process that includes:

- Finding out about your current kernel (what version it is and where it is located?)
- Obtaining the kernel (where do you get the kernel source, how do you unpack it and where do you put it?)
- Obtaining and reading documentation (where can I find out about my new kernel source?)
- Configuring your kernel (how is this done, what is this doing?)
- Compiling your kernel (how do we do this?)
- Testing the kernel (why do we do this and how?)
- Installing the kernel (how do we do this?)

But to begin with, we really need to look at exactly what the kernel physically is and how it is generated.

To do this, we will examine the Linux kernel, specifically on the x86 architecture.

The lifeless image

The kernel is physically a file that is usually located in the `/boot` directory. Under Linux, this file is called `vmlinuz`. On my system, an `ls` listing of the kernel produced:

```
bash# ls -al  /boot/vml*
lrwxrwxrwx   1 root     root           14 Jan  2 23:44 /boot/vmlinuz -> vmlinuz-2.0.31
-rw-r--r--   1 root     root       444595 Nov 10 02:59 /boot/vmlinuz-2.0.31
```

You can see in this instance that the “kernel file” is actually a link to another file containing the kernel image. The actual kernel size will vary from machine to machine. The reason for this is that the size of the kernel is dependant on what features you have compiled into it, what modifications you've make to the kernel data structures and what (if any) additions you have made to the kernel code.

`vmlinuz` is referred to as the kernel image. At a physical level, this file consists of a small section of machine code followed by a compressed block. At boot time, the program at the start of the kernel is loaded into memory at which point it uncompresses the rest of the kernel.

This is an ingenious way of making the physical kernel image on disk as small as possible; uncompressed the kernel image could be around one megabyte.

So what makes up this kernel?

Kernel gizzards

An umcompressed kernel is really a giant object file; the product of C and assembler linking - the kernel is not an "executable" file (i.e. you just can't type `vmlinuz` at the prompt to run the kernel). The actual source of the kernel is stored in the `/usr/src/linux` directory; a typical listing may produce:

```
[jamiesob@pug jamiesob]$ ls -al /usr/src
total 4
drwxr-xr-x   4 root     root         1024 Jan  2 23:53 .
drwxr-xr-x  18 root     root         1024 Jan  2 23:45 ..
lrwxrwxrwx   1 root     root           12 Jan  2 23:44 linux -> linux-2.0.31
drwxr-xr-x   3 root     root         1024 Jan  2 23:44 linux-2.0.31
drwxr-xr-x   7 root     root         1024 Jan  2 23:53 redhat
```

> `/usr/src/linux` is a soft link to `/usr/src/<whatever linux version> -` this means you can store several kernel source trees - however - you **MUST** change the soft link of `/usr/src/linux` to the version of the kernel you will be compiling as there are several components of the kernel source that rely on this.

SPECIAL NOTE: If your system doesn't have a `/usr/src/linux` or a `/usr/src/linux*` directory (where * is the version of the Linux source) or there is a /usr/src/linux directory but it only has a couple of files then you don't have the source code installed on your machine. The quick solution is to installed the RPM file containing the kernel source code from the Redhat CD-ROM. You might want to download a more recent version from the Internet.

If you are unsure about how to install a RPM file please refer to the Redhat guides.

A typical listing of `/usr/src/linux` produces:

```
-rw-r--r--   1 root     root            2 May 12  1996 .version
-rw-r--r--   1 root     root         6282 Aug  9  1994 CHANGES
-rw-r--r--   1 root     root        18458 Dec  1  1993 COPYING
-rw-r--r--   1 root     root        21861 Aug 17  1995 CREDITS
-rw-r--r--   1 root     root         3221 Dec 30  1994 Configure
-rw-r--r--   1 root     root         2869 Jan 10  1995 MAGIC
-rw-r--r--   1 root     root         7042 Aug 17  1995 Makefile
-rw-r--r--   1 root     root         9817 Aug 17  1995 README
-rw-r--r--   1 root     root         3114 Aug 17  1995 README.modules
-rw-r--r--   1 root     root        89712 May 12  1996 System.map
drwxr-xr-x   6 root     root         1024 May 10  1996 arch/
drwxr-xr-x   7 root     root         1024 May 10  1996 drivers/
drwxr-xr-x  13 root     root         1024 May 12  1996 fs/
drwxr-xr-x   9 root     root         1024 May 12  1996 include/
drwxr-xr-x   2 root     root         1024 May 12  1996 init/
drwxr-xr-x   2 root     root         1024 May 12  1996 ipc/
drwxr-xr-x   2 root     root         1024 May 12  1996 kernel/
drwxr-xr-x   2 root     root         1024 May 12  1996 lib/
drwxr-xr-x   2 root     root         1024 May 12  1996 mm/
drwxr-xr-x   2 root     root         1024 Jan 23  1995 modules/
drwxr-xr-x   4 root     root         1024 May 12  1996 net/
-rw-r--r--   1 root     root          862 Aug 17  1995 versions.mk
-rwxr-xr-x   1 root     root       995060 May 12  1996 vmlinux
```

Take note of the `vmlinux (if you have one)` file - this is the uncompressed kernel! Notice the size? [`vmlinuz` is the `.z` (or compressed) version of `vmlinux` plus the decompression code]

Within this directory hierarchy are in excess of 1300 files and directories. On my system this consists of around 400 C source code files, 370 C header files, 40 Assembler source files and 46 Makefiles. These, when compiled, produce around 300 object files and libraries. At a rough estimate, this consumes around 16 megabytes of space (this figure will vary).

While this may seem like quite a bit of code, much of it actually isn't used in the kernel. Quite a large portion of this is driver code; only drivers that are needed on the system are compiled into the kernel, and then only those that are required at run time (the rest can be placed separately in things called modules; we will examine this later).

The various directories form logical divisions of the code, especially between the architecture dependant code (`linux/arch`), drivers (`linux/drivers`) and architecture independent code. By using `grep` and `find`, it is possible to trace the structure of the kernel program, look at the boot process and find out how various parts of it work.

The first incision

An obvious place to start with any large C program is the `void main(void)` function. If you `grep` every source file in the Linux source hierarchy for this function name, you will be sadly disappointed.

As I pointed out earlier, the kernel is a giant object file - a series of compiled functions. It is NOT executable. The purpose of `void main(void)` in C is to establish a framework for the linker to insert code that is used by the operating system to load and run the program. This wouldn't be of any use for a kernel - it is the operating system!

This poses a difficulty - how does an operating system run itself?

Making the heart beat...

In the case of Linux, the following steps are performed to boot the kernel:

- The boot loader program (e.g. `lilo`) starts by loading the `vmlinuz` from disk into memory, then starts the code executing.
- After the kernel image is decompressed, the actual kernel is started. This part of the code was produced from assembler source; it is totally machine specific. The code for this is located in the `/usr/src/linux/arch/i386/kernel/head.S` file. Technically at this point the kernel is running. This is the first process (0) and is called swapper. Swapper does some low level checks on the processor, memory and FPU availability, then places the system into protected mode. Paging is enabled.

- Interrupts are disabled (every one) though the interrupt table is set up for later use. The entire kernel is realigned in memory (post paging) and some of the basic memory management structures are created.
- At this point, a function called `start_kernel` is called. `start_kernel` is physically located in `/usr/src/linux/init/main.c` and is really the core kernel function - really the equivalent of the `void main(void). main.c` itself is virtually the root file for all other source and header files.
- Tests are run (the FPU bug in Pentium chip is identified amongst other checks including examinations on the DMA chip and bus architecture) and the BogoMip setting is established.
- `start_kernel` sets up the memory, interrupts and scheduling. In effect, the kernel has now has multi-tasking enabled. The console already has had several messages displayed to it.
- The kernel command line options are parsed (those passed in by the boot loader) and all embedded device driver modules are initialised.
- Further memory initialisations occur, socket/networking is started and further bug checks are performed.
- The final action performed by swapper is the first process creation with `fork` whereby the `init` program is launched. Swapper now enters an infinite idle loop.

It is interesting to note that as a linear program, the kernel has finished running! The timer interrupts are now set so that the scheduler can step in and pre-empt the running process. However, sections of the kernel will be periodically executed by other processes.

This is really a huge oversimplification of the kernel's structure, but it does give you the general idea of what it is, what it is made up of and how it loads.

Modules

A module is a dynamically loadable object file containing functions for interfacing with a particular device or performing particular tasks. The concept behind modules is simple; to make a kernel smaller (in memory), keep only the bare basics compiled into the kernel. When the kernel needs to use devices, let it load modules into memory. If it doesn't use the modules, let them be unloaded from memory.

This concept has also revolutionised the way in which kernels are compiled. No longer do you need to compile every device driver into the kernel; you can simply mark some as modules. This also allows for separate module compilation - if a new device driver is released then it is a simple case of recompiling the module instead of the entire kernel.

Modules work by the kernel communicating with a program called `kerneld`. `kerneld` is run at boot time just like a normal daemon process. When the kernel notices that a request has come in for the use of a module, it checks if it is loaded in memory. If it is, then the routine is run, however, if not, the kernel gets `kerneld` to load the module into memory. `kerneld` also removes the

module from memory if it hasn't been used in a certain period of time (configurable).

The concept of modules is a good one, but there are some things you should be aware of:

- Frequently used devices and devices required in the boot process (like the hard disk) should **not** be used as modules; these must be compiled into the kernel.
- While the concept of modules is great for systems with limited memory, should you use them? Memory is cheap - compiling an object into the kernel rather than leaving it as a module may use more memory but is that better than a system that uses its CPU and IO resources to constantly load and unload modules? There are trade offs between smaller kernels and CPU/IO usage with loadable modules.
- It is probably a good idea to modularise devices like the floppy disk, CD-ROM and parallel port - these are not used very often, and when they are, only for a short time.
- It is NOT a good idea to modularise frequently used modules like those which control networking.

There is quite a bit more to kernel modules.

The proc file system

Part of the kernel's function is to provide a file-based method of interaction with its internal data structures; it does this via the `/proc` virtual file system.

The `/proc` file system technically isn't a file system at all; it is in fact a window on the kernel's internal memory structures. Whenever you access the `/proc` file system, you are really accessing kernel memory.

So what does it do?

Effectively the `/proc` file system is providing an instant snapshot of the status of the system. This includes memory, CPU resources, network statistics and device information. This data can be used by programs to gather information about a system, an example of which is the `top` program. `top` scans through the `/proc` structures and is able to present the current memory, CPU and swap information, as given below:

```
7:12pm  up  9:40,  1 user,  load average: 0.00, 0.00, 0.10
34 processes: 33 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:  0.5% user,  0.9% system,  0.0% nice, 98.6% idle
Mem:  14940K av, 13736K used,  1204K free,  5172K shrd,  1920K buff
Swap: 18140K av,  2304K used, 15836K free

PID USER     PRI  NI SIZE  RES SHRD STAT %CPU %MEM  TIME COMMAND
789 jamiesob  19   0  102  480  484 R     1.1  3.2  0:01 top
 98 root      14   0 1723 2616  660 S     0.3 17.5 32:30 X :0
  1 root       1   0   56   56  212 S     0.0  0.3  0:00 init [5]
 84 jamiesob   1   0  125  316  436 S     0.0  2.1  0:00 -bash
 96 jamiesob   1   0   81  172  312 S     0.0  1.1  0:00 sh /usr/X11/bin/star
 45 root       1   0   45  232  328 S     0.0  1.5  0:00 /usr/sbin/crond -l10
  6 root       1   0   27   72  256 S     0.0  0.4  0:00 (update)
  7 root       1   0   27  112  284 S     0.0  0.7  0:00 update (bdflush)
 59 root       1   0   53  176  272 S     0.0  1.1  0:00 /usr/sbin/syslogd
 61 root       1   0   40  144  264 S     0.0  0.9  0:00 /usr/sbin/klogd
```

```
   63 bin        1   0   60    0  188 SW    0.0  0.0  0:00 (rpc.portmap)
   65 root       1   0   58    0  180 SW    0.0  0.0  0:00 (inetd)
   67 root       1   0   31    0  180 SW    0.0  0.0  0:00 (lpd)
   73 root       1   0   84    0  208 SW    0.0  0.0  0:00 (rpc.nfsd)
   77 root       1   0  107  220  296 S     0.0  1.4  0:00 sendmail:accepting
```

The actual contents of the `/proc` file system on my system look like:

```
psyche:~$ ls /proc
1/          339/        7/          87/         dma          modules
100/        45/         71/         88/         filesystems  net/
105/        451/        73/         89/         interrupts   pci
108/        59/         77/         90/         ioports      self/
109/        6/          793/        96/         kcore        stat
116/        61/         80/         97/         kmsg         uptime
117/        63/         84/         98/         ksyms        version
124/        65/         85/         cpuinfo     loadavg
338/        67/         86/         devices     meminfo
```

Each of the numbered directories store state information of the process by their PID. The `self/` directory contains information for the process that is viewing the `/proc` filesystem, i.e. - YOU. The information stored in this directory looks like:

```
cmdline                 (Current command line)
cwd - [0303]:132247    (Link to the current working directory)
environ                 (All environment variables)
exe - [0303]:109739    (Currently executing code)
fd/                     (Directory containing virtual links to
                         file handles)
maps|                   (Memory map structure)
root - [0303]:2        (Link to root directory)
stat                    (Current process statistics)
statm                   (Current memory statistics)
```

Most of these files can be `cat`'ed to the screen. The `/proc/filesystems` file, when `cat`'ed, lists the supported file systems. The `/proc/cpuinfo` file gives information about the hardware of the system:

```
psyche:~$ cat /proc/cpuinfo
cpu             : 586
model           : Pentium 90/100
mask            : E
vid             : GenuineIntel
fdiv_bug        : no
math            : yes
hlt             : yes
wp              : yes
Integrated NPU  : yes
Enhanced VM86   : yes
IO Breakpoints  : yes
4MB Pages       : yes
TS Counters     : yes
Pentium MSR     : yes
Mach. Ch. Exep. : yes
CMPXCHGB8B      : yes
BogoMips        : 39.94
```

Be aware that upgrading the kernel may mean changes to the structure of the `/proc` file system. This may require software upgrades. Information about this should be provided in the kernel README files.

Exercises

13.1.	Find out where `kerneld` is launched from.
13.2.	What is the purpose of `/sbin/lsmod`? Try it. What is the contents of the file /proc/modules?
13.3.	Find out where your kernel image is located and how large it is.
13.4.	Examine the `/proc` file system on you computer. What do you think the `/proc/kcore` file is? Hint: Have a look at the size of the file.

Really, why bother?

The most common reason to recompile the kernel is because you've added some hardware and you want the kernel to recognise and (if you're lucky) use it. A very good time to recompile your kernel is after you've installed Linux. The reason for this is that the original Linux kernel provided has extra drivers compiled into it which consume memory. Funnily enough, while the kernel includes a driver for communicating in EBCDIC via a 300 baud modem to a coke machine sitting in the South Hungarian embassy in Cairo [Makefile Question:

```
Do you want to include support for coke machines located in Cairo?
[Y],N,M?
Do you want to support South Hungarian Embassy Models [Y],N,M?
Support for 300 baud serial link [Y],N,M?
Support EBCDIC communication[Y],N,M?
```

(I might be making this up... :)]

...the kernel, by default, doesn't have support for some very common sound cards and network devices! To be fair, there are good reasons for this (IRQ conflicts etc.) but this does mean a kernel recompile is required.

Another good reason to modify the kernel is to customise some of its data structures for your system. Possible modifications include increasing the number of processes the kernel can support (this is a fixed array and can't be set on run time) or modifying the size of certain buffers.

One of the great benefits of having the source code for the operating system is that you can play OS-Engineer; it is possible for you to change the scheduling algorithm, memory management scheme or the IPC functionality.

> While it might be nice to go and do these things, it would be unadvisable to modify the API if you want your programs to still run under Linux. However, there is nothing to stop you adding to the API. You may, for example, wish to add a system call to print "`Hello World`" to the screen (this would obviously be of great benefit to the rest of the Linux community ;) - this is possible for you to do.

Strangely enough, to modify the kernel, you need kernel source code. The actual source can be obtained from a variety of locations. For users who installed Linux from CD ROM, the source can be found within the distribution.

Typically you will actually go back into the installation menu and install **only** the section that contains the source.

However, more often than not, you are actually seeking to upgrade the kernel, so you need the latest kernel source. Because the development of the Linux kernel is an on-going process, new versions of development kernels are constantly being released. It is not unusual for development kernels to be released as often as once per day!

The Kernel HOWTO describes some ways to obtain kernels:

```
You can obtain the source via anonymous ftp from ftp.kernel.org in
/pub/linux/kernel/vx.y, where x.y is the version(eg 2.2), and as
mentioned before, the ones that end with an odd number are
development releases and may be unstable. It is typically labelled
linux-x.y.z.tar.gz, where x.y.z is the version number. The sites also
typically carry ones with a suffix of .bz2, which have been
compressed with bzip2 (these files will be smaller and take less time
to transfer).

It's best to use ftp.xx.kernel.org where xx is your country code;
examples being ftp.at.kernel.org for Austria, and ftp.us.kernel.org
for the United States.
```

Generally you will only want to obtain a "stable" kernel version. Kernels with even minor numbers are the stable kernels. Kernels with odd minor numbers in the version are the development kernels. My current kernel is 2.2.12-20. The minor number in the version number is 2 (the second one) indicating a stable production kernel.

The 2.3.X range of kernels is where people are working on new features for the Linux kernel.

As of writing this section (January 12, 1999) the latest versions are 2.2.14 for the stable version and 2.3.39 for the development branch. I need to upgrade the kernel on my machine.

> If you have an extremely new type of hardware then you are often forced into using developmental kernels. There is nothing wrong with using these kernels, but beware that you may encounter system crashes and potential losses of data. During a one year period, the author obtained around twenty developmental kernels, installed them and had very few problems. For critical systems, it is better to stick to known stable kernels.

RedHat and other companies which distribute versions of Linux make the latest kernel sources available in distribution specific formats. For example, RedHat primarily uses the RPM package manager.

So, you've obtained the kernel source - it will be in one large, compressed file. You know have to unpack the archive. If you are unsure the Linux Kernel HOWTO provides some guidance as does the Redhat guides.

A couple of points to note.

- *Some* sources install to directories given by the kernel version, not to the `linux` directory. It may be worth checking on this before you unpack the

source by issuing the following command. It will list all the files and directories that are contained in the `source_filename`, the kernel archive.

```
tar -txvf source_filename
```

- This will display a list of files and where they are to be installed. If they are to be installed into a directory other than `linux` then you must make a symbolic link, called `linux` in the `/usr/src` directory to the directory that contains the new source.
- **NEVER** just delete your old source - you may need it to recompile your old kernel version if you find the new version isn't working out, though we will discuss other ways round this problem in later sections.

If you are upgrading your kernel regularly, an alternative to constantly obtaining the complete kernel source is to patch your kernel.

Patches are basically text files that contain a list of differences between two files. A kernel patch is a file that contains the differences between all files in one version of the kernel to the next.

Why would you use them? The only real reason is to reduce download time and space. A compressed kernel source can be extremely large whereas patches are relatively small.

Patches are produced as the output from the `diff` command. For example, given two files:

```
file1

"vi is a highly exciting program with a wide range of great features – I am sure that
we will adopt it as part of our PlayPen suite"
        - Anonymous Multimillionaire Software Farmer

file2

"vi is a mildly useless program with a wide range of missing features – I am sure that
we will write a much better product; we'll call it `Sentence'"
          - Anonymous Multimillionaire Software Farmer
```

After executing the command:

```
diff file1 file2  file3
```

`file2` would contain:

```
1,2c1,2
< "vi is a highly exciting program with a wide range of great features - I
< am sure that we will adopt it as part of our PlayPen suite"
---
"vi is a mildly useless program with a wide range of missing features - I
am sure that we will write a much better product; we'll call it `Sentence'"
```

To apply a patch, you use the `patch` command. `patch` expects a file as a parameter to apply the patch to, with the actual patch file as standard input. Following the previous example, to patch `file1` with `file3` to obtain `file2`, we'd use the following command:

```
patch file1 < file3
```

This command applies the `file3` patch to `file1`. After the command, `file1` is the same as `file2` and a file called `file1.orig` has been created as a backup of the original `file1`.

The Linux HOWTO further explains applying a kernel patch:

```
 Incremental upgrades of the kernel are distributed as patches. For example,
if you have version 1.1.45, and you notice that there's a 'patch46.gz' out
there for it, it means you can upgrade to version 1.1.46 through application
of the patch. You might want to make a backup of the source tree first
('make clean' and then 'cd /usr/src; tar zcvf old-tree.tar.gz linux' will
make a compressed tar archive for you.).

So, continuing with the example above, let's suppose that you have
'patch46.gz' in /usr/src. cd to /usr/src and do a
'zcat patch46.gz | patch -p0' (or 'patch -p0 < patch46'
if the patch isn't compressed). You'll see things whizz by(or flutter by, if
your system is that slow) telling you that it is trying to apply hunks, and
whether it succeeds or not. Usually, this action goes by too quickly for you
to read, and you're not too sure whether it worked or not, so you might want
to use the -s flag to patch, which tells patch to only report error messages
(you don't get as much of the ''hey, my computer is actually doing something
for a change!'' feeling, but you may prefer this..). To look for parts which
might not have gone smoothly, cd to /usr/src/linux and look for files with a
.rej extension. Some versions of patch (older versions which may have been
compiled with on an inferior filesystem) leave the rejects with a #
extension. You can use 'find' to look for you;

    find .  -name '*.rej' -print

prints all files who live in the current directory or any subdirectories
with a .rej extension to the standard output.
```

Patches can be obtained from the same sites as the complete kernel sources.

A couple of notes about patches:

- For every new version of the kernel, there is a patch. To upgrade from a kernel version that is five versions behind the version you want, yo have to obtain and apply five patches (e.g. kernel `n.n.1` upgrading to `n.n.6` requires patches: patch2, patch3, patch4, patch5 and patch6). This gets tedious and is often easier and quicker to simply obtain the entire kernel source again.
- Patches are forever - when you patch your kernel source, you modify it for good.

Documentation

Every version of the kernel source comes with documentation. There are several "main" files you should read about your current source version including:

- `/usr/src/linux/README`
  Instructions on how to compile the kernel
- `/usr/src/linux/MAINTAINERS`
  A list of people who maintain the code
- `/usr/src/linux/Documentation/*`
  Documentation for parts of the kernel.

ALWAYS read the documentation after obtaining the source code for a new kernel, and especially if you are going to be compiling in a new kind of device.

The Linux Kernel-HOWTO is essential reading for anything relating to compiling or modifying the kernel.

Linux is the collaborative product of many people. This is something you quickly discover when examining the source code. The code (in general) is neat but sparsely commented; those comments that do exist can be absolutely riotous...well, at least strange :)

These are just a selection of the quotes found in the `/usr/src/linux/kernel` directory:

```
(fork.c)

        Fork is rather simple, once you get the hang of it, but the memory
        management can be a bitch.

(exit.c)

        "I ask you, have you ever known what it is to be an orphan?"

(module.c)

        ... This feature will give you ample opportunities to get to know
        the taste of your foot when you stuff it into your mouth!!!

(schedule.c)

        The "confuse_gcc" goto is used only to get better assembly code..
        Dijkstra probably hates me.

        To understand this, you have to know who Dijkstra was - remember OS?

        ... disregard lost ticks for now.. We don't care enough.

(sys.c)

        OK, we have probably got enough memory - let it rip.

        This needs some heave checking ...
        I just haven't get the stomach for it. I also don't fully
        understand. Let somebody who does explain it.

(time.c)

        This is ugly, but preferable to the alternatives.  Bad, bad....

        ...This is revolting.
```

Apart from providing light entertainment, the kernel source comments are an important guide into the (often obscure) workings of the kernel.

Modifying the Kernel

The main reason for recompiling the kernel is to include support for new devices - to do this you simple have to go through the compile process and answer "Yes" to a few questions relating to the hardware you want. However, in some cases you may actually want to modify the way in which the kernel works, or, more likely, one of the data structures the kernel uses. This might sound a bit daunting, but with Linux this is a relatively simple process.

For example, the kernel maintains a statically-allocated array for holding a list of structures associated with each process running on the system. When all of

these structures are used, the system is unable to start any new processes. This limit is defined within the `tasks.h` file located in `/usr/src/linux/include/linux/` in the form of:

```
/*
* This is the maximum nr of tasks - change it if you need to
*/
#define NR_TASKS        512
#define MAX_TASKS_PER_USER (NR_TASKS/2)
#define MIN_TASKS_LEFT_FOR_ROOT 4
```

> While 512 tasks may seem a lot, on a multiuser system this limit is quickly exhausted. Remember that even without a single user logged on, a Linux system is running between 30 and 50 tasks. For each user login, you can (at peak periods) easily exceed 5 processes per user. Adding this to web server activity (some servers can be running in excess of one hundred processes devoted to processing incoming `http` requests), mail server, telnet, ftp and other network services, the 512 process limit is quickly reached.

Increasing `NR_TASKS` and recompiling the kernel will allow more processes to be run on the system - the downside to this is that more memory will be allocated to the kernel data area in the form of the increased number of task structures (leaving less memory for user programs).

Other areas you may wish to modify include buffer sizes, numbers of virtual terminals and memory structures. Most of these should be modifiable from the `.h` files found in the kernel source "`include`" directories.

There are, of course, those masochists (like myself) who can't help tinkering with the kernel code and "changing" things (a euphemism for wrecking a nice stable kernel). This isn't a bad thing (there is an entire team of kernel developers world-wide who spend quite a bit of time doing this) but you've got to be aware of the consequences - total system annihilation is one. However, if you feel confident in modifying kernel code, perhaps you should take a quick look at: /usr/src/linux/kernel/sched.c or /usr/src/linux/mm/memory.c

(actually, look at the code anyway). These are two of the most important files in the kernel source, the first, `sched.c` is responsible for task scheduling. The second, `memory.c` is responsible for memory allocation. Perhaps someone would like to modify `memory.c` so that when the kernel runs out of memory that the system simply doesn't just "hang" (just one of my personal gripes there... ;)

Compiling the source

As we will discuss in the next section, ALL changes to the kernel should be compiled and tested on DISK before the "new" kernel is installed on the system. The following section will explain how this is done.

- Obtain the source of the version before the latest kernel. Install the source in the appropriate directory.

- Obtain the patch for the latest kernel source and apply it to the source files you previously retrieved.

If you don't have Internet access, do the same thing but using the CD-ROM. Pick a version of the kernel source, install it, then patch it with the patch for the next version

- Find out how to generate a patch file based on the differences between more than one file - what is the command that would recursively generate a patch file from two directories? (These puns are getting very sad)

As you are aware (because you've read all the previous chapters and have been paying intense attention), `make` is a program use to compile source files, generate object files and link them. `make` actually lets the compilers do the work, however it co-ordinates things and takes care of dependencies. Important tip: Dependencies are conditions that exist due to that fact some actions have to be done after other actions - this is confusing, but wait, it gets worse. Dependencies also relate to the object of the action; in the case of `make` this relates to if the object (an object can be an object file or a source file) has been modified. For example, using our Humpty scenario:

`humpty` (program) is made up of legs, arms and torso (humpty, being an egg lacked a neck, thus his torso and head are one) - these could be equated to object files. Humpty's legs are made up of feet, shins and thighs - again, object files. Humpty's feet are made up of toes and other bits (how do you describe an egg's foot???) - these could be equated to source files. To construct `humpty`, you'd start at the simplest bits, like toes, and combine them with other bits to for the feet, then the legs, then finally, `humpty`.

You could not, however, fully assemble the leg without assembling the foot. And if you modified Humpty's toes, it doesn't mean you'd have to recompile his fingers - you'd have to reconstruct the foot object, relink into a new leg object, which you'd link with the (pre compiled and unmodified) arms and torso objects - thus forming Humpty.

`make`, while not specifically designed to handle broken egg reconstruction, does the same thing with source files - based entirely of rules which the user defines within a file called a `Makefile`. However, `make` is also clever enough to compile and link *only* the bits of a program that have been modified since the last compile.

In the case of the kernel, a series of `Makefiles` are responsible for the kernel construction. Apart from calling compilers and linkers, `make` can be used for running programs, and in the case of the kernel, one of the programs it calls is an initialisation script.

The steps to compile the kernel all make use of the `make` program. To compile the kernel, you must be in the `/usr/src/kernel`, and issue (in the following order and as the `root` user) these commands:

```
make config or make menuconfig or make xconfig
make dep
make clean
make zImage or make zdisk
make zlilo (if the previous was make zImage)
```

If you are going to be using modules with your kernel, you will require the following two steps:

```
make modules
make modules_install
```

The following is an explanation of each step.

Configuration

`make config` is the first phase of kernel recompilation. Essentially `make config` causes a series of questions to be issued to the user. These questions relate to what components should be compiled into the kernel. The following is a brief dialog from the first few questions prompted by `make config`:

```
psyche:~/usr/src/linux$ make config

rm -f include/asm
( cd include ; ln -sf asm-i386 asm)
/bin/sh scripts/Configure arch/i386/config.in
#
# Using defaults found in .config
#
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers
(CONFIG_EXPERIMENTAL)[N/y?] n
*
* Loadable module support
*
Enable loadable module support (CONFIG_MODULES) [Y/n/?] Y
Set version information on all symbols for modules
(CONFIG_MODVERSIONS)[N/y/?]
Kernel daemon support (e.g. autoload of modules) (CONFIG_KERNELD)
[N/y/?] y
*
* General setup
*
Kernel math emulation (CONFIG_MATH_EMULATION) [Y/n/?]
```

A couple of points to note:

- Each of these questions has an automatic default (capitalised). This default will be changed if you choose another option; i.e. If the default is "N" and you answer "Y" then on the next compile the default will be "Y". This means that you can simply press "enter" through most of the options after your first compile.
- These first few questions relate to the basic kernel setup: note the questions regarding modules. This is important to answer correctly, as if you wish to include loadable module support, you must do so at this point.

As you progress further through the questions, you will be prompted for choosing support for specific devices, for example:

```
*
* Additional Block Devices
*
Loopback device support (CONFIG_BLK_DEV_LOOP) [N/y/m/?]
```

```
Multiple devices driver support (CONFIG_BLK_DEV_MD) [N/y/?]
RAM disk support (CONFIG_BLK_DEV_RAM) [Y/m/n/?]
Initial RAM disk (initrd) support (CONFIG_BLK_DEV_INITRD) [N/y/?]
XT harddisk support (CONFIG_BLK_DEV_XD) [N/y/m/?]
```

In this case, note the "m" option? This specifies that the support for a device should be compiled in as a module - in other words, not compiled into the kernel but into separate modules.

Be aware that there are quite a few questions to answer in `make config`. If at any point you break from the program, you must start over again. Some "sections" of `make config`, like the sound card section, save the results of the first `make config` in a configuration file; you will be prompted to either reconfigure the sound card options or use the existing configurations file.

There are two other methods of configuring the kernel, `make menuconfig` and `make xconfig`.

The first time you run either of these configuration programs, they will actually be compiled before your very eyes (exciting eh?). `menuconfig` is just a text based menu where you select the parts of the kernel you want; `xconfig` is the same thing, just for X-Windows. Using either of these utilities will probably be useful for someone who has never compiled the kernel before, however, for a comprehensive step-by-step selection of kernel components, `make config` is, in my view, better. You may be wondering what is the result of `make config/menuconfig/xconfig`? What is actually happening is that small configuration files are being generated to be used in the next step of the process, `make dep`.

Dependencies

`make dep` takes the results from `make config` and "sets up" which parts of the kernel have to be compiled and which don't. Basically this step involves extensive use of `sed` and `awk` for string substitution on files. This process may take a few minutes; there is no user interaction at this point.

After running `make dep`, `make clean` must be run. Again, this process requires no user interaction. `make clean` actually goes through the source tree and removes all the old object and temporary files. **This process can not be skipped**.

At this point, we are ready to start the compile process.

Compilation

You have two options at this point; you may either install the kernel on the hard drive of the system and hope it works, or, install the kernel on a floppy disk and test it for a while, then (if it is working) install it on the hard drive.

ALWAYS tests your kernel on a floppy disk before installing it as your boot kernel on the hard drive. Why? Simply because if you install your new kernel directly over the one on the hard drive *and it doesn't work properly* (i.e.. crashes or hangs your system) then you will have difficulty booting your

system (being a well prepared Systems Administrator, you'd have a boot disk of course ... ;).

To compile your new kernel to disk, you must issue the command:

```
make zdisk
```

This will install a bootable kernel on the disk in A:. To boot the system, you simply insert the disk containing the kernel in A:, shut down the system, and let it reboot. The kernel on disk will load into memory, mount your root partition and the system will boot as normal. It is a good idea to run this kernel on disk for at least a few days, if not longer. If something goes wrong and you find your system has become unstable, it is merely a process of removing the disk, rebooting and the system will start up with your old kernel.

If you are going to install the kernel directly to the hard disk, then you should issue the commands:

```
make zImage
make zlilo
```

The first command, `make zImage`, actually compiles the kernel, the second, `make zlilo` installs the kernel on whatever root partition you have configured with lilo.

> Most systems use `lilo` as the kernel boot loader. A common misconception is that `lilo` is only used to boot kernels off hard disks. This is actually incorrect; if `lilo` is configured (usually done when you installed your system, see "`man lilo`" for more information on configuring it) to boot the kernel from floppy disk, then running `make zlilo` will cause a copy of the kernel (and lilo) to be copied onto a disk. However, `lilo` is usually used to load a kernel form hard disk. The way it works is simple; `lilo` finds the absolute block/sector address of the kernel image on the disk. It then creates a small program (containing this and other information) and inserts it in the boot sector of the primary hard disk. At boot time, `lilo` is run, prompting (optionally) the user for the desired operating system to boot. When the choice is made, `lilo` goes directly to the block/sector of the kernel boot image (or other operating system boot file) and loads it into memory and executes it.

The actual compile process (either using `make zImage` or `make zdisk` is a lengthy process. A Pentium 100 with 16 megabytes of RAM takes around 15 to 25 minutes to compile the kernel (depending on what has been included). Compiling DEC UNIX on a DEC-Alpha takes around three to four minutes. Have pity for those in the not-so-distant era of the 386 that waited all day for a kernel to recompile.

> It is quite OK to be recompiling the kernel while other users are logged onto the system; be aware that this will slow the process down and make the system appear VERY slow to the users (unless you have a "really, nice" machine).

If you have decided to use dynamically loadable modules, there are two more commands you must issue:

```
make modules
make modules_install
```

Note this is done *post kernel compile* - the useful thing about this is that if you upgrade your modules, you can simply recompile them without the need for a full kernel recompile!

After the `make zImage/zlilo/zdisk` commands and compiling the modules, your kernel is ready to be tested. As previously stated, it is important to test your kernel before using it as your system boot kernel.

If you find that the kernel is working normally from disk and it hasn't crashed the system (too much), then you can install the kernel to the hard disk. The easiest way to do this is to go back to the `/usr/src/linux` directory and type:

```
make zlilo
```

This will install the copy of the kernel that was previously compiled to disk (a copy is also kept in the kernel source directory) to the hard drive, or whatever boot device `lilo` is configured to use.

Common Problems

Did you read the documentation? "If all else fails, read the documentation" - this quote is especially true of kernel recompiles. A few common problems that you may be confronted with are:

- **`make` can not find the `Makefile` but it is there!:**
  This is because `make` is broken. This was a big problem under the `1.2.n` kernels when an updated `libc.so.x` library was released. The problem was that `make` would not work under `1.3.n` kernels that had been recompiled under the `1.2.n` versions with the new library; consequently, you couldn't recompile the kernel under the `1.3.n` kernels due to the fact `make` was not working! This has been fixed since, though at the time the solution was to go and get a new version of `make`. This is a classic example of what can happen when you start upgrading kernels without upgrading all the libraries, compilers and utilities. Always read the `README` file before recompiling the kernel and make sure you have all the right versions of libraries, compilers and utilities.

- **`make config/dep/clean` dies:**
  This is bad news. It means one of several things: either the config scripts can't find `/bin/bash` or `/bin/sh`, some of the source tree is missing, you are not running the program as `root` or there is something wrong with your system file permissions/links. It is very rare for this to happen with kernels "unpacked straight from the box". If it does happen, check for the previous reasons; if all else fails, go and get another kernel source.

- **`make zImage/zdisk` fails:**
  This is one of those sinking feeling moments when you start getting messages during the compile saying "Error: Something didn't

compile/link". Two primary reasons for this are: not running `make clean` after `make dep` and not having the correct libraries installed.

- **The kernel compiles and boots but it is unstable:**
  If you are using developmental kernels, this comes with the territory: because developmental kernels can be unstable. If, however, you are using a known "stable" kernel, then the reason is most likely a hardware conflict. Typical culprits are sound cards and network cards. Remove these from the kernel and recompile. You should then examine the documentation on the offending devices to see what the conflict is. Other reasons for kernel instability include compiling in support for devices you don't have (this is rare but can happen) or the fact that you've just discovered a "real" bug in the kernel - in which case the README documentation will assist you in locating the right person to talk to.

If you are still encountering problems, you should examine the newsgroup archives concerned with Linux. There are also several useful mailing lists and web sites that can assist you with kernel problems.

Exercises

13.5.	Modify the kernel so that the maximum number of tasks it can run is 50. Compile this kernel to a floppy disk. See how long it takes to use all these processes up.
13.6.	Modify your kernel so that the kernel version message (seen on boot time) contains your name. Hint: `/usr/src/linux/init` contains a file called `version.c` - modify a data structure in this.
13.7.	Recompile your own kernel, including only the components you need. For those components that you need but don't use very oftem, compile them in as modules. Initially boot the kernel from disk, then install it on your hard disk.

Conclusions

In this chapter we have examined:

- What is a kernel?
- Why would a Systems Administrator recompile a kernel?
- What makes up a modern kernel?
- How would you obtain a kernel?
- Why and how would you modify the kernel source?
- How is a kernel configured and recompiled?
- Why should a kernel be tested?
- How is a kernel installed?
- Issues associated with the modern Linux kernel

Further information of the Linux kernel can be obtained from the Linux Kernel HOWTO and the other resources mentioned at the beginning of this chapter.

Review Questions

13.1.

Describe the functions of the kernel; explain the difference between a kernel that uses modules and one that doesn't.

13.2.

You have added a D-Link ethernet card to your **laptop** (a D-Link ethernet card runs via the parallel port). Describe the steps you'd perform to allow the system to recognise it. Would you compile support for this module directly into the kernel or make it a module? Why/Why not?

13.3.

You wish to upgrade the kernel on an older system (ver 1.2.n) to the latest kernel. What issues should you consider? What problems could occur with such an upgrade; how would you deal with these?

Chapter 14

Automation and Observation

Introduction

Setting up a machine is one part of Systems Administration. Another somewhat more important part is keeping that machine going. Central to achieving this aim are the two activities we look at in this chapter (there are more)

1. Automation
   Any tasks which occurs more than once must be automated. The primary tool on UNIX systems for achieving this are shell programs. This chapter looks at the use of cron (the Linux scheduler) for automatically scheduling shell programs and other tasks.
2. Observation
   People will do things to your computer. Some of them will do nasty things. Observation is the act of keeping an eye on your machine so you know there is something wrong with it before the users complain about something not working.
   We look at observation from two perspectives: historical and current. Historical observation tells you what has happened on your system. Current observation tells you what is happening now.

Other Resources

Other resources which discuss similar topics include

- LAME
  A section called Automatic tasks with Cron and Crontab files.
- USAIL
  A section on automating tasks with Cron.
  http://www.uwsg.indiana.edu/usail/index/automate.html

Automation and `cron`

A number of the responsibilities of a System Administrator are automated tasks that must be carried out at the regular times every day, week or hour. Examples include, early every morning freeing up disk space by deleting entries in the `/tmp` directory, performing backups every night or compressing and archiving log files.

Most of these responsibilities require no human interaction other than to start the command. Rather than have the Administrator start these jobs manually,

UNIX provides a mechanism that will automatically carry out certain tasks at set times. This mechanism relies on the `cron` system.

For example, the mirror of the Linux Documentation Project (LDP) on the 85321 website is kept up to date with a cron job (a task scheduled with cron). This particular cron job, run every sunday night, connects to the central LDP site and transfers any updated data.

Components of `cron`

The cron system consists of the following three components

- `crontab` (the cron configuration) files
  These are the files which tell the cron system which tasks to perform and when.
- the `crontab` command
  This is the command used to modify the crontab files. Even though the crontab files are text files they should not be edited using a text editor.
- the daemon, `crond`
  The cron daemon is responsible for reading the crontab file and then performing the required tasks at the specified times. The cron daemon is started by a system startup file.

`crontab` format

`crontab` files are text files with each line consisting of 6 fields separated by spaces. The first five fields specify when to carry out the command and the sixth field specifies the command. Table 14.1, on the following page, outlines the purpose of each of the fields.

Field	Purpose
minute	minute of the hour, 00 to 59
hour	hour of the day, 00 to 32 (military time)
day	day of the month, 1 to 31
month	month of the year, 1 to 12
weekday	day of the week, Linux uses three letter abbreviations, sun, mon, tue,....
command	The actual command to execute

Table 14.1
`crontab` **fields**

Comments can be used and are indicated using the # symbol just as with shell programs. Anything that appears after a # symbol until the end of that line is considered a comment and is ignored by `crond`.

The five time fields can also use any one of the following formats

- an asterix that matches all possible values,
- a single integer that matches that exact value,

- a list of integers separated by commas (no spaces) used to match any one of the values
- two integers separated by a dash (a range) used to match any value within the range.

For example

Some example `crontab` entries include (all but the first two examples are taken from the Linux man page for `crontab`)

```
0 * * * * echo Cuckoo Cuckoo > /dev/console 2>&1
```

Every hour (when minutes=0) display Cuckoo Cuckoo on the system console.

```
30 9-17 * 1 sun,wed,sat echo ‘date‘ >> /date.file 2>&1
```

At half past the hour, between 9 and 5, for every day of January which is a Sunday, Wednesday or Saturday, append the date to the file `date.file`

```
0 */2 * * * date
```

Every two hours at the top of the hour run the `date` command

```
0 23-7/2,8 * * * date
```

Every two hours from 11p.m. to 7a.m., and at 8a.m.

```
0 11 4 * mon-wed date
```

At 11:00 a.m. on the 4th and on every mon, tue, wed

```
0 4 1 jan * date
```

4:00 a.m. on january 1st

```
0 4 1 jan * date >> /var/log/messages 2>&1
```

Once an hour, all output appended to log file

Output

When commands are executed by the `crond` daemon there is no terminal associated with the process. This means that standard output and standard error, which are usually set the terminal, must be redirected somewhere else. In this case the output is emailed to the person who's `crontab` file the command appears. It is possible to use I/O redirection to redirect the output of the commands to files. Some of the examples above use output redirection to send the output of the commands to a log file.

Exercises

14.1.	Write crontab entries for the following. - run the program `date` every minute of every day and send the output to a file called `date.log` - remove all the contents of the directory `/tmp` at 5:00am every morning - execute a shell script `/root/weekly.job` every Wednesday - run the program `/root/summary` at 3, 6 and 9 pm for the first five days of a month

Creating `crontab` files

`crontab` files should not be modified using an editor instead they should be created and modified using the `crontab` command. Refer for the manual page for `crontab` for more information but the following are two of the basic methods for using the command.

1. `crontab [file]`

2. `crontab [-e | -r | -l ] [username]`

Version 1 is used to replace an existing `crontab` file with the contents of standard input or the specified file.

Version 2 makes use of one of the following command line options

- `-e`
  Allows the user to edit the `crontab` file using an editor (the command will perform some additional actions to make it safe to do so)
- `-r`
  Remove the user's `crontab` file
- `-l`
  Display the user's `crontab` file onto standard output

By default all actions are carried out on the user's own `crontab` file. Only the root user can specify another username and modify that user's `crontab` file.

Exercise

14.2.	Using the `crontab` command to add the following to your `crontab` file and observe what happens. run the program `date` every minute of every day and send the output to a file called `date.log`

Current Observation

A part of the day to day operation of a system is keeping an eye on the systems current state. This section introduces a number of commands and tools that can be used to examine the current state of the system.

The tools are divided into two sections based on what they observe. The sections are

- disk and file system observation, and
  The commands `du` and `df`
- process observation and manipulation.
  The commands `ps`, `kill`, `nice` and `top`.

`df`

`df` summarises that amount of free disk space. By default `df` will display the following information for all mounted file systems

- total number of disk blocks,
- number of disk blocks used,
- number available
- percentage of disk blocks used, and
- where the file system is mounted.

`df` also has an option, `-i` to display Inode usage rather than disk block usage. What an Inode is will be explained in a later chapter. Simply every file that is created must have an Inode. If all the Inodes are used you can't create anymore files. Even if you have disk space available.

The `-T` option will cause `df` to display each file systems type.

Exercise

14.3.	Use the `df` command to answer the following questions - how many partitions do you have mounted - how much disk space do you have left on your Linux partition - how many more files can you create on your Linux partition

`du`

The `du` command is used to discover the amount of disk space used by file or directory. By default `du` reports file size as a number of 1 kilobyte blocks. There are options to modify the command so it reports size in bytes (`-b`) or kilobytes (`-k`).

If you use `du` on a directory it will report back the size of each file and directory within it and recursively descend down any sub-directories. The `-s` switch is used to produce the total amount of disk used by the contents of a directory.

There are other options that allow you to modify the operation of `du` with respect to partitions and links.

Exercise

14.4.	Use the `du` command to answer the following questions - how many blocks does the `/etc/passwd` file use, - how large (in bytes) is the `/etc/passwd` file, - how disk space is used by the `/etc/` directory, the `usr` directory

System Status

Table 14.2 summarises some of the commands that can be used to examine the current state of your machine. Some of the information they display includes

- amount of free and used memory,
- the amount of time the system has been up,
- the load average of the system,
  Load average is the number processes ready to be run and is used to give some idea of how busy your system is.
- the number of processes and amount of resources they are consuming.

Some of the commands are explained below. For those that aren't use your system's manual pages to discover more.

Command	Purpose
`free`	display the amount of free and used memory
`uptime`	how long has the system been running and what is the current load average
`Ps/pstree`	one off snap shot of the current processes
`top`	continual listing of current processes
`uname`	display system information including the hostname, operating system and version and current date and time
`gtop`	The Gnome system monitor, a GUI which provides a view of running processes, memory and file system usage (see chapter 5)

Table 14.2
System status commands

`ps`

The `ps` command displays a list of information about the process that were running at the time the `ps` command was executed.

`ps` has a number of options that modify what information it displays. Table 14.3 lists some of the more useful or interesting options that the Linux version of PS supports.

Table 14.4 explains the headings used by `ps` for the columns it produces.

For more information on the `ps` command you should refer to the manual page.

Option	Purpose
l	long format
u	displays username (rather than uid) and the start time of the process
m	display process memory info
a	display processes owned by other users (by default ps only shows your processes)
x	shows processes that aren't controlled by a terminal
f	use a tree format to show parent/child relationships between processes
w	don't truncate lines to fit on screen

Table 14.3
ps options

Field	Purpose
NI	the nice value
SIZE	memory size of the processes code, data and stack
RSS	kilobytes of the program in memory (the resident set size)
STAT	the status of the process (R-runnable, S-sleeping, D-uninterruptable sleep, T-stopped, Z-zombie)
TTY	the controlling terminal

Table 14.4
ps fields

Exercise

14.5. Use the ps command to answer the following questions
- how many processes do you currently own
- how many processes are running on your system
- how much RAM does the ps command use
- what's the current running process

top

ps provides a one-off snap shot of the processes on your system. For an on-going look at the processes Linux generally comes with the top command. It also displays a collection of other information about the state of your system including

- uptime, the amount of time the system has been up
- the load average,
- the total number of processes,
- percentage of CPU time in user and system mode,

- memory usage statistics
- statistics on swap memory usage

Refer to the man page for `top` for more information.

`top` is not a standard UNIX command however it is generally portable and available for most platforms.

`top` displays the process on your system ranked in order from the most CPU intensive down and updates that display at regular intervals. It also provides an interface by which you can manipulate the nice value and send processes signals.

The nice value

The nice value specifies how "nice" your process is being to the other users of the system. It provides the system with some indication of how important the process is. The lower the nice value the higher the priority. Under Linux the nice value ranges from -20 to 19.

By default a new process inherits the nice value of its parent. The owner of the process can increase the nice value but cannot lower it (give it a higher priority). The root account has complete freedom in setting the nice value.

`nice`

The `nice` command is used to set the nice value of a process when it first starts.

`renice`

The `renice` command is used to change the nice value of a process once it has started.

Signals

When you hit the `CTRL-C` combination to stop the execution of a process a signal (the TERM signal) is sent to the process. By default many processes will terminate when they receive this signal

The UNIX operating system generates a number of different signals. Each signal has an associated unique identifying number and a symbolic name. Table 14.6 lists some of the more useful signals used by the Linux operating system. There are 32 in total and they are listed in the file `/usr/include/linux/signal.h`

Chapter 5 has some additional discussion about signals.

`SIGHUP`

The `SIGHUP` signal is often used when reconfiguring a daemon. Most daemons will only read the configuration file when they startup. If you modify the configuration file for the daemon you have to force it to re-read the file. One method is to send the daemon the `SIGHUP` signal.

SIGKILL

This is the big "don't argue" signal. Almost all processes when receiving this signal will terminate. It is possible for some processes to ignore this signal but only after getting themselves into serious problems. The only way to get rid of these processes is to reboot the system.

Symbolic Name	Numeric identifier	Purpose
SIGHUP	1	hangup
SIGKILL	9	the kill signal
SIGTERM	15	software termination

Table 14.5
Linux signals

kill

The `kill` command is used to send signals to processes. The format of the `kill` command is

`kill [-`*`signal`*`] `*`pid`*

This will send the signal specified by the number *`signal`* to the process identified with process identifier *`pid`*. The kill command will handle a list of process identifiers and signals specified using either their symbolic or numeric formats.

By default kill sends signal number 15 (the TERM signal).

Historical Observation

There will be times when you want to reconstruct what happened in the lead up to a problem. Situations where this might be desirable include

- you believe someone has broken into your system,
- one of the users performed an illegal action while online, and
- the machine crashed mysteriously at some odd time.
- You want to track how much a particular system or resource is used, e.g. a Web server. This can also be useful in justifying to management the need for additional resources.

This is where

- logging, and
  The recording of certain events, errors, emergencies.
- accounting.
  Recording who did what and when.

become useful.

This section examines the methods under Linux by which logging and accounting are performed. In particular it will examine

- the `syslog` system,
- process accounting, and
- login accounting.

Managing log and accounting files

Both logging and accounting tend to generate a great deal of information especially on a busy system. One of the decisions the Systems Administrator must make is what to do with these files. Options include

- don't create them in the first place,
  The head in the sand approach. Not a good idea.
- keep them for a few days, then delete them, and
  If a problem hasn't been identified within a few days then assume there is no reasons to keep the log files. Therefore delete the existing ones and start from scratch.
- keep them for a set time and then archive them.
  Archiving these files might include compressing them and storing them online or copying them to tape.

`logrotate`

Linux systems come with a command called logrotate. As the name suggests this command is used to aid in the management of log files. `logrotate` allows the automatic rotation, compression, removal and mailing of log files on a daily, weekly, monthly or size basis. On Redhat Linux the logrotate command is configured with the file /etc/logrotate.conf.

Centralise

If you are managing multiple computers it is advisable to centralise the logging and accounting files so that they all appear on the one machine. This makes maintaining and observing the files easier. The syslog system (discussed below) provides this ability.

Security

Since log files are your record of what has occured it is important that they are stored securely. This is another reason for keeping the log files for computers on a single, very secure system. One of the first things someone breaking into your system will attempt to do is to modify the log files so that their actions don't appear.

Keeping log files safe is especially important as in some situations they may be required as legal evidence.

Look at them

Late in 1999 the disk drives in the computer which acts as the Web server for a certain faculty at a certain University failed. It appears the RAID controller for the disk had detected and started logging errors with the disk about five months earlier. The problem was that no-one was reading the log file. It is important that log files actually be read.

Logging

The ability to log error messages or the actions carried out by a program or script is fairly standard. On earlier versions of UNIX each individual program would have its own configuration file that controlled where and what to log. This led to multiple configuration and log files that made it difficult for the Systems Administrator to control and each program had to know how to log.

`syslog`

The `syslog` system was devised to provide a central logging facility that could be used by all programs. This was useful because Systems Administrators could control where and what should be logged by modifying a single configuration file and because it provided a standard mechanism by which programs could log information.

Components of `syslog`

The `syslog` system can be divided into a number of components

- default log file,
  On many systems messages are logged by default into the file `/var/log/messages`
- the `syslog` message format,
- the application programmer's interface,
  The API programs use to log information.
- the daemon, and
  The program that directs logging information to the correct location based on the configuration file.
- the configuration file.
  Controls what information is logged and where it is logged.

Exercise

14.6.	Examine the contents of the file `/var/log/messages`. You will probably have to be the root user to do so. One useful piece of information you should find in that file is a copy of the text that appears as Linux boots.

`syslog` message format

`syslog` uses a standard message format for all information that is logged. This format includes

- a facility,
  The facility is used to describe the part of the system that is generating the message. Table 14.3 lists some of the common facilities.
- a level,
  The level indicates the severity of the message. In lowest to highest order the levels are `debug info notice warning err crit alert emerg`
- and a string of characters containing a message.

Facility	Source
`kern`	the kernel
`mail`	the mail system
`lpr`	the print system
`daemon`	a variety of system daemons
`auth`	the login authentication system

Table 14.6
Common `syslog` facilities

`syslog`'s API

In order for `syslog` to be useful application programs must be able to pass messages to the `syslog` daemon so it can log the messages according to the configuration file.. There are at least two methods which application programs can use to send messages to `syslog`. These are:

- `logger`,
  `logger` is a UNIX command. It is designed to be used by shell programs which wish to use the `syslog` facility.
- the `syslog` API.
  The API (application program interface) consists of a set of the functions (`openlog syslog closelog`) which are used by programs written in compiled languages such as C and C++. This API is defined in the `syslog.h` file. You will find this file in the system include directory `/usr/include`.

Exercises

14.7. Examine the manual page for `logger`. Use `logger` from the command line to send a message to `syslog`

14.8. Examine the manual page for `openlog` and write a C program to send a message to `syslog`

`syslogd`

`syslogd` is the `syslog` daemon. It is started when the system boots by one of the startup scripts. `syslogd` reads its configuration file when it startups or when it receives the `HUP` signal. The standard configuration file is `/etc/syslog.conf`.

`syslogd` receives logging messages and carries out actions as specified in the configuration file. Standard actions include

- appending the message to a specific file,
- forwarding the message to the `syslogd` on a different machine, or
- display the message on the consoles of all or some of the logged in users.

`/etc/syslog.conf`

By default `syslogd` uses the file `/etc/syslog.conf` as its configuration file. It is possible using a command line parameter of `syslogd` to use another configuration file.

A `syslog` configuration file is a text file. Each line is divided into two fields separated by one or more spaces or tab characters

- a selector, and
  Used to match log messages.
- an action.
  Specifies what to do with a message if it is matched by the selector

The selector

The selector format is `facility.level` where `facility` and level `level` match those terms introduced in the `syslog` message format section from above.

A selector field can include

- multiple selectors separated by `;` characters
- multiple facilities, separated by a `,` character, for a single level
- an `*` character to match all facilities or levels

The level can be specified with or without a `=`. If the `=` is used only messages at exactly that level will be matched. Without the `=` all messages at or above the specified level will be matched.

`syslog.conf` actions

The actions in the `syslog` configuration file can take one of four formats

- a pathname starting with `/`
  Messages are appended onto the end of the file.
- a hostname starting with a `@`
  Messages are forwarded to the `syslogd` on that machine.

- a list of users separated by commas
  Messages appear on the screens of those users if they are logged in.
- an asterix
  Messages are displayed on the screens of all logged in users.

For example

The following is an example `syslog` configuration file taken from the Linux manual page for `syslog.conf`

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                         /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none              /var/log/messages

# The authpriv file has restricted access.
authpriv.*                      /var/log/secure

# Log all the mail messages in one place.
mail.*                          /var/log/maillog

# Everybody gets emergency messages, plus log them on another
# machine.
*.emerg                         *

# Save mail and news errors of level err and higher in a
# special file.
uucp,news.crit                      /var/log/spooler
```

Exercise

14.9. A common problem on many systems are users who consume too much disk space. One method to deal with this is to have a script which regularly checks on disk usage by users and reports those users who are consuming too much. The following is one example of a script to do this.

```
#!/bin/bash

# global constant
# DISKHOGFILE holds the location of the file defining each users
# maximum disk space
DISKHOGFILE="disk.hog"
# OFFENDERFILE specifiesl where to write information about offending
# users
OFFENDERFILE="offender"

space_used()
  # accept a username as 1st parameter
  # return amount of disk space used by the users home directory
  # in a variable usage
{
  # home directory is the sixth field in /etc/passwd
  the_home=‘grep ^$1: /etc/passwd | cut -d: -f6‘
  # du uses a tab character to seperate out its fields
  # we’re only interested in the first one
```

```
  usage='du -s $the_home | cut -f1'
}

#
# Main Program
#

while read username max_space
do
  space_used $username
  if [ $usage -gt $max_space ]
  then
    echo $username has a limit of $max_space and has used $used  $OFFENDERFILE
  fi
done < $DISKHOGFILE
```

Modify this script so that it uses the `syslog` system rather than displaying its output onto standard output.

14.10. Configure syslog so the messages from the script in the previous question are appended to the logfile `/var/log/disk.hog.messages` and also to the main system console.

Accounting

Accounting was developed when computers were expensive resources and people were charged per command or CPU time. In today's era of cheap, powerful computers its rarely used for these purposes. One thing accounting is used for is as a source of records about the use of the system. Particular useful if someone is trying, or has, broken into your system.

In this section we will examine

- login accounting.
  Keeping a track of who has logged into the system and how long they were logged in.
- process accounting

Login accounting

The file `/var/log/wtmp` is used to store the username, terminal port, login and logout times of every connection to a Linux machine. Every time you login or logout the `wtmp` file is updated. This task is performed by `init`.

`last`

The `last` command is used to view the contents of the `wtmp` file. There are options to limit interest to a particular user or terminal port.

Exercise

14.11. Use the `last` command to
- count how many logins there have been since the current `wtmp` file was created,
- how many times has the root user logged in

ac

The last command provides rather rudimentary summary of the information in the `wtmp` file. As a Systems Administrator it is possible that you may require more detailed summaries of this information. For example, you may desire to know the total number of hours each user has been logged in, how long per day and various other information.

The command that provides this information is the `ac` command.

Installing ac

It is possible (even likely) that you will not have the `ac` command installed. The ac command is part of the psacct package. You may have to install it. Refer to the RedHat guides for information on how to do this..

14.12.	Use the ac command to - find the total number of hours you were logged in as the root user - find the average number of hours per login for all users - find the daily totals for root

Process accounting

Also known as CPU accounting, process accounting records the elapsed CPU time, average memory use, I/O summary, the name of the user who ran the process, the command name and the time each process finished. You may also need to install process accounting.

Turning process accounting on

Process accounting does not occur until it is turned on using the `accton` command.

```
accton /var/log/acct
```

Where `/var/log/acct` is the file in which the process accounting information will be stored. The file must already exist before it will work. You can use any filename you wish but many of the accounting utilities rely on you using this file.

lastcomm

`lastcomm` is used to display the list of commands executed either for everyone, for particular users, from particular terminals or just information about a particular command. Refer to the `lastcomm` manual page for more information.

```
[root@beldin /proc]# lastcomm david
netscape               david    tty1       0.02 secs Sun Jan 25 16:26
[root@beldin /proc]# lastcomm ttyp2
lastcomm               root     ttyp2      0.55 secs Sun Jan 25 16:21
ls                     root     ttyp2      0.03 secs Sun Jan 25 16:21
ls                     root     ttyp2      0.02 secs Sun Jan 25 16:21
accton                 root     ttyp2      0.01 secs Sun Jan 25 16:21
```

The `sa` command

The `sa` command is used to provide more detailed summaries of the information stored by process accounting and also to summarise the information into other files.

```
[root@beldin /proc]# /usr/sbin/sa -a
      66       0.19re       0.25cp
       6       0.01re       0.16cp   cat
       8       0.00re       0.04cp   lastcomm
      17       0.00re       0.01cp   ls
       6       0.01re       0.01cp   man
       1       0.00re       0.01cp   troff
       5       0.01re       0.01cp   less
       1       0.15re       0.01cp   in.ftpd
       6       0.01re       0.01cp   sh
       5       0.00re       0.00cp   gunzip
       1       0.00re       0.00cp   grotty
       2       0.00re       0.00cp   sa
       1       0.00re       0.00cp   groff
       1       0.00re       0.00cp   gtbl
       1       0.00re       0.00cp   gzip
       1       0.00re       0.00cp   sh*
       1       0.00re       0.00cp   netscape*
       1       0.00re       0.00cp   accton
       2       0.00re       0.00cp   bash*
```

Refer to the manual pages for the `sa` command for more information.

So what?

This section has given a very brief overview of process and login accounting and the associated commands and files. What use do these systems fulfil for a Systems Administrator? The main one is that they allow you to track what is occurring on your system and who is doing it. This can be useful for a number of reasons

- tracking which user's are abusing the system
- figuring out what is normal for a user
  If you know that most of your users never use commands like sendmail and the C compilers (via process accounting) and then all of a sudden they start using this might be an indication of a break in.
- justifying to management the need for a larger system
  Generally management won't buy you a bigger computer just because you want one. In most situations you will have to put together a case to justify why the additional expenditure is necessary. Process and login account could provide some of the necessary information.

Conclusions

The `cron` system is used to automatically perform tasks at set times. Components of the `cron` system include

- the daemon, `crond`,
  Which actually performs the specified tasks.
- `crontab` files, and
  That specify the when and what.
- the `crontab` command.
  Used to manipulate the `crontab` files.

Useful commands for examining the current status of your systems file system include `df` and `du`. Commands for examining and manipulating processes include `ps`, `kill`, `renice`, `nice` and `top`. Other "status" commands include `free`, `uptime` and `uname`.

`syslog` is a centralised system for logging information about system events. It's components include

- an API and a program (`logger`) by which information can be logged,
- the `syslogd` daemon that actually performs the logging, and
- the `/etc/syslog.conf` that specifies what and where logging information should be logged.

Login accounting is used to track when, where and for how long users connect to your system. Process accounting is used to track when and what commands were executed. By default Linux does not provide full support for either form of accounting (it does offer some standard login accounting but not the extra command `sac`). However there are freely available software distributions that provide Linux this functionality.

Login accounting is performed in the `/var/log/wtmp` file that is used to store the details of every login and logout from the system. The `last` command can be used to view the contents of the binary `/var/log/wtmp` file. The non-standard command `sac` can be used to summarise this information into a number of useful formats.

Process accounting must be turned on using the `accton` command and the results can be viewed using the `lastcomm` command.

Both logging and accounting can produce files that grow to some considerable size in a short amount of time. The Systems Adminstrator must implement strategies to deal with these log files. Either by ignoring and deleting them or by saving them to tape.

Review Questions

14.1

Explain the relationship between each of the following

- `crond`, `crontab` files and the `crontab` command,
- `syslogd`, `logger` and `/etc/syslog.conf`
- `/var/adm/wtmp`, `last` and `sac`

14.2

You have just modified the `/etc/syslog.conf` file. Will your changes take effect immediately? If not what command would you use to make the modifications take effect? How could you check that the modifications are working?

14.3

Write crontab entries to achieve the following

- run the script `/usr/local/adm/bin/archiveIt` every Monday at 6 am
- run a script `/usr/local/adm/bin/diskhog` on Monday, Wednesday and Friday at 6am, 12pm, 4pm

14.4

You are a script kiddie (a deogatory name for someone who breaks into computers using very simplistic and automated approaches) who has just broken into my Linux computer. You need to answer the following

- How do you find what form of logging and accounting I have installed?
- What can you do to cover the fact that you have broken into my system?

Chapter 15

Networks: The Connection

Introduction

Connecting computers to networks and managing those networks are probably the most important, or at least the most hyped, areas of computing at the moment. This and the following chapter introduce the general concepts associated with TCP/IP-based networks and in particular the knowledge required to connect and use Linux computers to those networks.

This chapter examines how you connect a Linux machine and configure it to provide basic network connections and services for other machines. Higher level network applications, such as file sharing and Web servers, and how they work and what you can do with them is the topic for the following chapter.

This chapter

- Overview
  Provides an overview of connecting a Linux machine to a network.
- TCP/IP Basics
  A brief introduction to the fundamentals of TCP/IP networking.
- Hardare
  Quick coverage of the hardware which can be used to networking
- Kernel support
- Network configuration

Other Resources

As you might expect there is a large amount of information about creating and maintaining TCP/IP networks on the Internet. The following is a small list of some of that material

- HOWTOs
  Linux Networking-HOWTO which describes how to install and configure the Linux networking software and associated tools. Linux Networking Overview HOWTO provides an overview of the networking capabilities of Linux and provides pointers to further information. Multicast over TCP/IP HOWTO, DNS HOWTO covers the configuration of the Domain Name Service on Linux, Ethernet HOWTO, IPX HOWTO covers the installation on Linux of the network protocol used by Novell, IP Masquerade HOWTO, ISP Hookup HOWTO, PLIP Install HOWTO covers how to connect Linux boxes using null parallel cables, PPP HOWTO, Asymmetric Digital Subscriber Loop mini-HOWTO, Bridge mini-HOWTO,

Bridge+Firewall mini-HOWTO, Cipe+Masquerading mini-HOWTO, IP Alias mini-HOWTO, IP Subnetworking mini-HOWTO, Leased Line mini-HOWTO, Token Ring mini-HOWTO, VPN mini-HOWTO, Linux Modem Sharing mini-HOWTO

- LDP Guides
  The Linux Installation and Getting Started Guide's Chapter 6 covers networking. The major one is the Linux Network Administrators Guide. It is a touch old (March 1996) but it was actually published by O'Rielly and Associates (http://www.ora.com/) but is also freely available as part of the Linux Documentation Project.
- Linux network project
  Development on the Linux networking code is an on-going project. The project leader maintains a Web site which contains information about the current developments. It's located at
  http://www.uk.linux.org/NetNews.html
- comp.os.linux.networking
  A newsgroup specifically for discussions about Linux networking.
- TCP/IP introduction and administration,
  Documents produced by Rutgers University. Available from `ftp://athos.rutgers.edu/runet/` with the filenames `tcp-ip-intro` and `tcp-ip-admin` as either Word documents or postscript files. Should also be present on the 85321 website/CD-ROM
- RFC Database
  RFCs (Request for comments) are the standards documents for the Internet. A Web-based interface to the collection of RFCs is available from `http://pubweb.nexor.co.uk/public/rfc/index/rfc.html`
- Linux for an ISP
  A number of Internet Service Providers from throughout the world use Linux servers. There is a Web page which maintains a list of of links of interest to these folk. It is available at
  `http://www.anime.net/linuxisp/` Some of the links are dated.

The Overview

This chapter introduces the process and knowledge for connecting a Linux machine to a TCP/IP network. There are many other types of networking protocols but TCP/IP is the protocol family on the Internet so that is the one we concentrate on.

Creating a TCP/IP network does not necessarily mean you are connected to the Internet. You can have a TCP/IP network between the two computers you have at home.

What you need

In order to create some sort of TCP/IP network using Linux you will need the following

- Networking hardware
  You will need to make some sort of connection between the machines on your network so they can communicate. Linux supports a wide range of networking hardware. You can only use networking hardware Linux supports (unless you want to start writing device drivers).
- Appropriately configured kernel
  To use your network hardware the kernel must contain the appropriate device driver or have access to an appropriate module. The kernel also requires a number of other components which provide necessary low-level support for networking. If you are using some sort of strange hardware you may need to recompile the kernel to include support for your hardware.
- Network configuration tools
  These should be already present on most Linux systems and are used to configure networking.
- Network applications
  These are the topic of the next chapter and again most are supplied with the common Linux distributions. These provide the higher level services such as email, Web and file sharing.
- Network information
  This information is necessary to configure your system on the network. It includes your machine's IP address, the network address, the broadcast and netmask addresses, the router address and the address of your DNS server.

What you do

To install your Linux box onto a network you move on up the layers with steps something like the following

- Obtain the appropriate hardware
- Connect it to your system
- Configure your kernel to recognise the hardware
- Configure the network software
- Test the connection

TCP/IP Basics

Before going any further it is necessary to introduce some of the basic concepts related to TCP/IP networks. An understanding of these concepts is essential for the the next steps in connecting a Linux machine to a network. If you find the following too confusing or disjointed please refer to some of the other resources mentioned at the start of this chapter. The concepts introduced in the following includes

- hostnames
  Every machine (also known as a host) on the Internet has a name. This section introduces hostnames and related concepts.

- IP addresses
  Each network interface on the network also has a unique IP address. This section discusses IP addresses, the components of an IP address, subnets, network classes and other related issues.
- Name resolution
  Human beings use hostnames while the IP protocols use IP addresses. There must be a wy, name resolution, to convert hostnames into IP addresses. This section looks at how this is achieved.
- Routing
  When network packets travel from your computer to a Web site in the United States there are normally a multitude of different paths that packet can take. The decisions about which path it takes are performed by a routing algorithm. This section briefly discusses how routing occurs.

Hostnames

Most computers on a TCP/IP network are given a name, usually known as a host name (a computer can be known as a host). The hostname is usually a simple name used to uniquely identify a computer within a given site. A fully qualified Internet host name, also known as a fully qualified domain name (FQDNFQDN), uses the following format

```
hostname.site.domain.country
```

- `hostname`
  A name by which the computer is known. This name must be unique to the site on which the machine is located.
- `site`
  A short name given to the site (company, University, government department etc) on which the machine resides.
- `domain`
  Each site belongs to a specific domain. A domain is used to group sites of similar purpose together. Table 15.? provides an example of some domain names. Strictly speaking a domain name also includes the country.
- `country`
  Specifies the actual country in which the machine resides. Table 15.? provides an example of some country names. You can see a list of the country codes at http://www.bcpl.net/~jspath/isocodes.html

For example the CQU machine `jasper`'s fully qualified name is `jasper.cqu.edu.au`, where `jasper` is the hostname, `cqu` is the site name, the domain is `edu` and the country is `au`.

Domain	Purpose
edu	Educational institution, university or school
com	Commercial company
gov	Government department
net	Networking companies

Table 15.1
Example Internet domains

Country code	Country
nothing or us	United States
au	Australia
uk	United Kingdom
In	India
Ca	Canada
Fr	France

Table 15.2
Example Country Codes

hostname

Under Linux the hostname of a machine is set using the `hostname` command. Only the root user can set the hostname. Any other user can use the `hostname` command to view the machine's current name.

```
root@faile david]# hostname
faile.cqu.edu.au
[root@faile david]# hostname fred
[root@faile david]# hostname
fred
```

Changes to the hostname performed using the `hostname` command will not apply after you reboot a RedHat Linux computer. RedHat Linux sets the hostname during startup from one of its configuration files, `/etc/sysconfig/network` This is the file which is changed by the GUI tools provided with RedHat. If you wish a change in hostname to be retained after you reboot you will have to change this file.

Qualified names

`jasper.cqu.edu.au` is a fully qualified domain name and uniquely identifies the machine `jasper` on the CQU campus to the entire Internet. There cannot be another machine called `jasper` at CQU. However there could be another machine called `jasper` at James Cook University in Townsville (its fully qualified name would be `jasper.jcu.edu.au`).

A fully qualified name must be unique to the entire Internet. Which implies every hostname on a site should be unique.

Not qualified

It is not always necessary to specify a fully qualified name. If a user on `aldur.cqu.edu.au` enters the command `telnet jasper` the networking software assumes that because it isn't fully qualified hostname the user means the machine `jasper` on the current site (`cqu.edu.au`).

IP/Internet Addresses

Alpha-numeric names, like hostnames, cannot be handled efficiently by computers, at least not as efficiently as numbers. For this reason, hostnames are only used for us humans. The computers and other equipment involved in TCP/IP networks use numbers to identify hosts on the Internet. These numbers are called IP addresses. This is because it is the Internet Protocol (IP) which provides the addressing scheme.

IP addresses are currently 32 bit numbers, IPv6 the next generation of IP uses 128 bit address. IP addresses are usually written as four numbers separated by full stops (called dotted decimal form) e.g. `132.22.42.1`. Since IP addresses are 32 bit numbers, each of the numbers in the dotted decimal form are restricted to between 0-255 (32 bits divide by 4 numbers gives 8 bits per number and 255 is the biggest number you can represent using 8 bits). This means that `257.33.33.22` is an invalid address.

Dotted Quad to Binary

The address `132.22.42.1` in dotted decimal form is actually stored on the computer as `10000100 00010110 00101010 00000001`. Each of the four decimal numbers represent one byte of the final binary number

- 132 = 10000100
- 22 = 00010110
- 42 = 00101010
- 1 = 00000001

The convervsion from dotted quad to binary (and back again) is important for some of the following concepts.

Networks and hosts

An IP address actually consists of two parts

- a network portion, and
  This is used to identify the network that the machine belongs to. Hosts on the same network will have this portion of the IP address in common. This is one of the reasons why IP masquerading is reqired for mobile computers (e.g. laptops). If you move a computer to a different network you must

give it a different IP address which includes the network address of the new network it is connected to.

- the host portion.
  This is the part which uniquely identifies the host on the network.

The network portion of the address forms the high part of the address (the bit that appears on the left hand side of the number). **The size of the network and host portions of an IP address is specified by another 32 bit number called the netmask (also known as the subnet mask)netmask.**

To calculate which part of an IP address is the network and which the host the IP address and the subnet mask are treated as binary numbers (see diagram 15.?). Each bit of the subnet mask and the IP address are compared and

- if the bit is set in both the IP address and the subnet mask then the bit is set in the network address,
- if the bit is set in the IP address but **not** set in the subnet mask then the bit is set in the host address.

For example

```
IP address  138.77.37.21       10001010  01001101  00100101  00100101
netmask     255.255.255.0   11111111  11111111  11111111  00000000

network address  138.77.37.0  10001010  01001101  00100101  00000000
host address    0.0.0.21     00000000  00000000  00000000  00100101
```

The Internet is a network of networks

The structure of IP addresses can give you some idea of how the Internet works. It is a network of networks. You start with a collection of machines all connected via the same networking hardware, a local area network. All the machines on this local area network will have the same network address, each machine also has a unique host address.

The Internet is formed by connecting a lot of local area networks together.

For example

In Figure 15.5 there are two networks, `138.77.37.0` and `138.77.36.0`. These are two networks on the Rockhampton campus of Central Queensland University and both use ethernet as their networking hardware. This means that when a computer on the 37 subnet (the network with the network address 138.77.37.0) wants to send information to another computing on the 37 subnet it simply uses the characteristics of ethernet. The information is placed on the ethernet network and gets broadcasted to every ethernet card on the network. The ethernet card which has the appropriate address is the only one which "accepts" the information.

However, if the machine 138.77.37.37 wants to send information to the machine 138.77.36.15 it's a bit more complex. Since both computers are on separate networks the machine `138.77.37.37` just can't send information to the machine `138.77.36.15`. Instead it has to use a gateway machine (only

rarely is the gateway machine a computer but it can be). The gateway machine actually has two network connections. One connection to the `138.77.37.0` network and the other to the `138.77.36.0` network.

It is via this dual connection that the gateway acts as the connection between the two networks. The gateway knows that it should grab any and all packets on the `138.77.36.0` network destined for the `138.77.37.0` network (and vice versa). When it grabs these packets the gateway machine transfers them from the network device connected to the sending network to the network device connected to the receiving network.

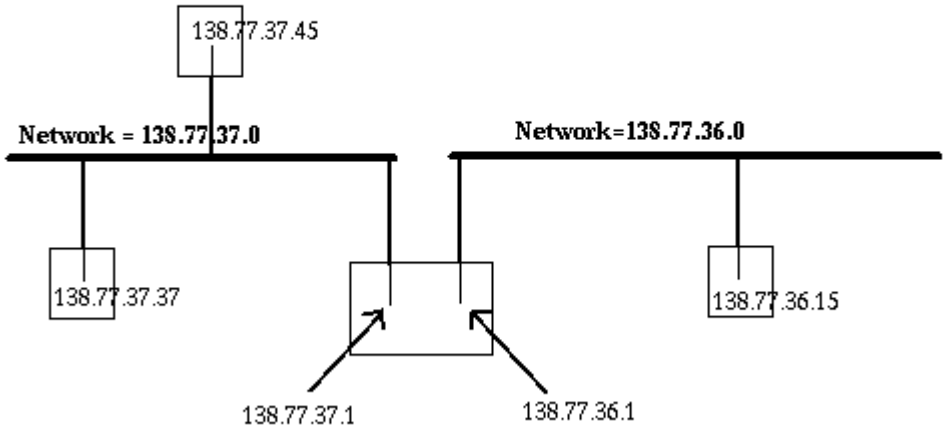


Figure 15.5
A simple gateway

This process is repeated for other networks. Each network is then connected to each other via devices called routers, or perhaps gateways. This is a very simple example.

Assigning IP addresses

Some IP addresses are reserved for specific purposes and you should not assign these addresses to a machine. Table 15.3 lists some of these addresses

Address	Purpose
`xx.xx.xx.0`	network address
`xx.xx.xx.1`	gateway address *
`xx.xx.xx.255`	broadcast address
`127.0.0.1`	loopback address

*** this is not a set standard**

Table 15.3
Reserved IP addresses

As mentioned above 127.0.0.1 is a special IP address. It refers to the local host (or the loopback address). The local host allows software to address the local machine in exactly the same way it would address a remote machine. For those of you without network connections the localhost will be the only method you can use to experiment with the concepts introduced in this and the following chapter.

As shown in the previous examples gateways and routers are able to distribute data from one network to another because they are actually physically connected to two or more networks through a number of network interfaces. Figure 15.5 provides a representation of this.

The machine in the middle, the gateway machine, has two network interfaces. One has the IP address `138.77.37.1` and the other `138.77.36.1` (it's common practice for a networks gateway machine to have the host id 1, but by no means compulsory).

By convention the network address is the IP address with a host address that is all 0's. The network address is used to identify a network.

The broadcast address is the IP address with the host address set to all 1's and is used to send information to all the computers on a network, typically used for routing and error information.

Network Classes

During the development of the TCP/IP protocol stack IP addresses were divided into classes. There are three main address classes, A, B and C. Table 15.? summarises the differences between the three classes. The class of an IP address can be deduced by the value of the first byte of the address.

Class	First byte value	Netmask	Number of hosts
A	1 to 126	255.0.0.0	16 million
B	128 to 191	255.255.0.0	64,000
C	192 to 223	255.255.255.0	254
Multicast	224 – 239	240.0.0.0	

Table 15.4
Network classes

If you plan on setting up a network that is connected to the Interet the addresses for your network must be allocated to you by central controlling organisation. You can't just choose any set of addresses you wish, chances are they are already taken my some other site.

If your network will not be connected to the Internet you can choose from a range of addresses which have been set aside for this purpose. These addresses are shown in Table 15.4

Network class	Addresses
A	10.0.0.0 to 10.255.255.255
B	172.16.0.0 to 172.31.255.255
C	192.168.0.0 to 192.168.255.255

Table 15.5
Networks reserved for private networks

Subnets

Central Queensland University has a class B network address, `138.77.0.0`. This would imply that you could make the following assumptions about the IP address `138.77.1.1`. The network address is `138.77.0.0` and that the host address is `1.1`, this is after all how a class B address is defined.

If you did make these implications you would be wrong.

CQU has decided to break its available IP addresses into further networks, called subnets. Subnetting works by moving the dividing line between the network address bits and the host address bits. Instead of using the first two bytes for the network address CQU uses subnetting to use the first three bytes. This is achieved by setting the netmask to `255.255.255.0`.

This means that the address `138.77.1.1` actually breaks up into a network address `138.77.1.0` and a host address of `1`. The network `138.77.1.0` is said to be a subnet of the larger `138.77.0.0` network.

Why subnet?

Subnetting is used for a number of reasons including

- security reasons,
  Using ethernet all hosts on the same network can see all the packets on the network. So it makes sense to put the computers in student labs on a different network to the computer on which student results are placed.
- physical reasons,
  Networking hardware, like ethernet, has physical limitations. You can't put machines on the Mackay campus on the same network as machines on the Rockhampton campus (they are separated by about 300 kilometers).
- political reasons, and
  There may be departments or groups within an organisation that have unique needs or want to control their own network. This can be achieved by subnetting and allocating them their own network.
- hardware and software differences.
  Someone may wish to use completely different networking hardware and software.

"Strange" subnets

Generally subnet masks are byte oriented, for example `255.255.255.0`. This means that divide between the network portion of the address and the host portion occurs on a byte boundary. However it is possible and sometimes necessary to use bit oriented subnet masks, for example `255.255.255.224`. Bit oriented implies that this division occurs within a byte.

For example a small company with a class C Internet address might use the subnet mask `255.255.255.224`.

Exercises

15.1.	Complete the following table by calculating the network and host addresses. (refer back to the example earlier in the chapter)

IP address	Subnet mask	Network address	Host address
`178.86.11.1`	`255.255.255.0`		
`230.167.16.132`	`255.255.255.192`		
`132.95.132.5`	`255.255.240.0`		

Name resolution

We have a problem. People will use hostnames to identify individual computers on the network while the computers use the IP address. How are the two reconciled.

When you enter `http://www.lycos.com/` on your WWW browser the first thing the networking software must do is find the IP address for `www.lycos.com`. Once it has the IP address it can connect to that machine and download the WWW pages.

The process of taking a hostname and finding the IP address is called **name resolution**.

Methods of name resolution

There are two methods that can be used to perform name resolution

- the `/etc/hosts` file, and
- the Domain Name Service.

`/etc/hosts`

One way of performing name resolution is to maintain a file that contains a list of hostnames and their equivalent IP addresses. Then when you want to know a machine's IP address you look up the file.

Under UNIX the file is `/etc/hosts`. `/etc/hosts` is a text file with one line per host. Each line has the format

```
IP_address hostname aliases
```

Comments can be indicated by using the hash # symbol. Aliases are used to indicate shorter names or other names used to refer to the same host.

For example

For example the hosts file of the machine `aldur` looks like this

```
# every machine has the localhost entry
127.0.0.1       localhost         loopback
138.77.36.29    aldur.cqu.edu.au  aldur
138.77.1.1      jasper.cqu.edu.au jasper
138.77.37.28    pol.cqu.edu.au    pol
```

Problems with `/etc/hosts`

When a user on `aldur` enters the command `telnet jasper.cqu.edu.au` the software first looks in the hosts file for an entry for `jasper`. If it finds an entry it obtains jasper's IP address and then can execute the command.

What happens if the user enters the command `telnet knuth`. There isn't an entry for `knuth` in the hosts file. This means the IP address of `knuth` can't be found and so the command can't succeed.

One solution would be to add an entry in the hosts file for every machine the users of `aldur` wish to access. With over two million machines on the Internet it should be obvious that this is not a smart solution.

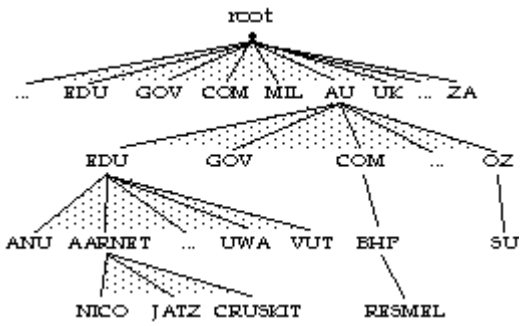
Domain name service (DNS)

The following reading on the DNS was taken from `http://www.aunic.net/dns.html`

In the early days of the Internet, all host names and their associated IP addresses were recorded in a single file called hosts.txt, maintained by the Network Information Centre in the USA. Not surprisingly, as the Internet grew so did this file, and by the mid-80's it had become impractically large to distribute to all systems over the network, and impossible to keep up to date. The Internet Domain Name System (DNS) was developed as a distributed database to solve this problem. It's primary goal is to allow the allocation of host names to be distributed amongst multiple naming authorities, rather than centralised at a single point.

DNS structure

The DNS is arranged as a hierarchy, both from the perspective of the structure of the names maintained within the DNS, and in terms of the delegation of naming authorities. At the top of the hierarchy is the root domain "." which is administered by the Internet Assigned Numbers Authority (IANA). Administration of the root domain gives the IANA the authority to allocate domains beneath the root, as shown in the diagram below:



The process of assigning a domain to an organisational entity is called delegating, and involves the administrator of a domain creating a sub-domain and assigning the authority for allocating sub-domains of the new domain the subdomain's administrative entity.

This is a hierarchical delegation, which commences at the "root" of the Domain Name Space ("."). A fully qualified domain name, is obtained by writing the simple names obtained by tracing the DNS hierarchy from the leaf nodes to the root, from left to right, separating each name with a stop ".", eg.

```
fred.xxxx.edu.au
```

is the name of a host system (huxley) within the XXXX University (xxx), an educational (edu) institution within Australia (au).

The sub-domains of the root are known as the top-level domains, and include the edu (educational), gov (government), and com (commercial) domains. Although an organisation anywhere in the world can register beneath these three-character top level domains, the vast majority that have are located within, or have parent companies based in, the United States. The top-level domains represented by the ISO two-character country codes are used in most other countries, thus organisations in Australia are registered beneath au.

The majority of country domains are sub-divided into organisational-type sub-domains. In some countries two character sub-domains are created (eg. ac.nz for New Zealand academic organisations), and in others three character sub-domains are used (eg. com.au for Australian commercial organisations). Regardless of the standard adopted each domain may be delegated to a separate authority.

Organisations that wish to register a domain name, even if they do not plan to establish an Internet connection in the immediate short term, should contact the administrator of the domain which most closely describes their activities.

Even though the DNS supports many levels of sub-domains, delegations should only be made where there is a requirement for an organisation or organisational sub-division to manage their own name space. Any sub-domain administrator must also demonstrate they have the technical competence to operate a domain name server (described below), or arrange for another organisation to do so on their behalf.

Domain Name Servers

The DNS is implemented as collection of inter-communicating nameservers. At any given level of the DNS hierarchy, a nameserver for a domain has knowledge of all the immediate sub-domains of that domain.

For each domain there is a primary nameserver, which contains authoritative information regarding Internet entities within that domain. In addition Secondary nameservers can be configured, which periodically download authoritative data from the primary server. Secondary nameservers provide backup to the primary nameserver when it is not operational, and further improve the overall performance of the DNS, since the nameservers of a domain that respond to queries most quickly are used in preference to any others.

`/etc/resolv.conf`

When performing a name resolution most UNIX machines will check their `/etc/hosts` first and then check with their name server. How does the machine know where its domain name server is. The answer is in the `/etc/resolv.conf` file.

`resolv.conf` is a text file with three main types of entries

- # comments
  Anything after a # is a comment and ignored.
- domain *name*
  Defines the default domain. This default domain will be appended to any hostname that does not contain a dot.
- nameserver *address*
  This defines the IP address of the machines domain name server. It is possible to have multiple name servers defined and they will be queried in order (useful if one goes down).

For example

The `/etc/resolv.conf` file from my machine is listed below.

```
domain cqu.edu.au
nameserver 138.77.5.6
nameserver 138.77.1.1
```

Routing

So far we've looked at names and addresses that specify the location of a host on the Internet. We now move onto routing. Routing is the act of deciding how each individual datagram finds its way through the multiple different paths to its destination.

Simple routing

For most UNIX computers the routing decisions they must make are simple. If the datagram is for a host on the local network then the data is placed on the

local network and delivered to the destination host. If the destination host is on a remote network then the datagram will be forwarded to the local gateway. The local gateway will then pass it on further.

However, a network the size of the Internet cannot be constructed with such a simple approach. There are portions of the Internet where routing is a much more complex business, too complex to be covered as a portion of one week of a third year unit.

Routing tables

Routing is concerned with finding the right **network** for a datagram. Once the right network has been found the datagram can be delivered to the host.

Most hosts (and gateways) on the Internet maintain a routing table. The entries in the routing table contain the information to know where to send datagrams for a particular network.

Constructing the routing table

The routing table can be constructed in one of two ways

- constructed by the Systems Administrator, sometimes referred to as static routes,
- dynamically created by a number of different available routing protocols

The dynamic creation by routing protocols is complex and beyond the scope of this subject.

Exercises

15.2. Why is the name server in `/etc/resolv.conf` specified using an IP address and not a hostname?

TCP/IP Basics Conclusion

The Internet is a network of networks. Each network has its own network address. Each computer on those networks has its own network address. Network addresses are allocated in classes. You can't simply choose an IP address yourself. It must match the network you are connecting to and not be used by anyone else. Most organisations with a range of IP address will split them into subnets.

Software and hardware use IP addresses to identify computers. People use hostnames. Name resolution makes the connection between a hostname and an IP address (and vice versa). On a small scale name resolution can be done with a local file. However, scaling to a large network requires the use of the Domain Name Service.

Routing is the act of delivering packets of information to the appropriate place. With a single physical network routing is quite straight forward. However with a large network of networks maintaining the rules about the routes from one to another network can get quite complex.

Network Hardware

The first step in connecting a machine to a network is to find out what sort of network hardware you will be using. The aim of this unit and this chapter is not to give you a detailed introduction to networking hardware. If you are interested in the topic there are a number of readings and resources mentioned throughout this section.

Before you can use a paticular type of networking hardware, or any hardware for that matter, there must be support for that device in the Linux kernel. If the kernel doesn't support the required hardware then you can't use it. Currently the Linux kernel offers support for the networking hardware outlined in list below. For more detailed information about hardware support under Linux refer to the Hardware Compatibility HOWTO available from your nearest mirror of the Linux Documentation Project.

Some of the hardware supported includes arcnet, ATM http://lrcwww.epfl.ch/linux-atm/, AX25 amateur radio, FDDI, Frame relay, ISDN, modems, serial and parallel, radio modem, token ring, X.25, WaveLan, wireless, card, and ethernet

In most "normal" situations the networking hardware being used will be either

- modem
  A modem is a serial device so your Linux kernel should support the appropriate serial port you have in your computer. The networking protocol used on a modem will be either SLIP or PPP which must also be supported by the kernel.
- ethernet
  Possibly the most common form of networking hardware at the moment. There are a number of different ethernet cards. You will need to make sure that the kernel supports the particular ethernet card you will be using. The Hardware Compatibility HOW-TO and the Ethernet HOWTO cover this information.

Network devices

As mentioned in chapter 10 the only way a program can gain access to a physical device is via a device file. Network hardware is still hardware so it follows that there should be device files for networking hardware. Under other versions of the UNIX operating system this is true. It is not the case under the Linux operating system.

Device files for networking hardware are created, as necessary, by the device drivers contained in the Linux kernel (ethernet and others) or by user programs which make network connections (e.g. modems, PPP connections). These device files are not available for other programs to use. This means I can't execute the command

```
cat < /etc/passwd > /dev/eth0
```

The only way information can be sent via the network is by going through the kernel.

Remember, the main reason UNIX uses device files is to provide an abstraction which is independent of the actual hardware being used. A network device file must be configured properly before you can use it send and receive information from the network. The process for configuring a network is discussed later in this chapter.

The installation process for RedHat will normally perform some network configuration for you. Tofind out what network devices are currently active on your system have a look at the contents of the file `/proc/net/dev`/

```
[david@faile]$ cat /proc/net/dev
Inter-|   Receive                |  Transmit
 face |packets errs drop fifo frame|packets errs drop fifo colls carrier
    lo:     91    0    0    0    0       91    0    0    0     0    0
  eth0:      0    0    0    0    0       60    0    0    0     0   60
```

On this machine there are two active network devices. `lo:` the loopback device and `eth0:` an ethernet device file. If a computer has more than one ethernet interface (network devices are usually called network interfaces) you would normally see entries for eth1 eth2 etc.

IP aliasing (talked about more later) is the ability for a single ethernet card to have more than one Internet address (often used when a single computer is acting as the Web server for many different sites). The following example shows the contents of the /proc/net/dev file for a machine using IP aliasing. *It is not normal for an ethernet card to have multiple IP addresses, normally each ethernet card/interface will have one IP address*.

```
[david@cq-pan ]$ cat /proc/net/dev
Inter-|   Receive                |  Transmit
face |packets errs drop fifo frame|packets errs drop fifo colls carrier
  lo: 285968    0    0    0    0   285968    0    0    0     0    0
eth0:61181891  59   59    0   89 77721923    0    0    0 11133617   57
eth0:0:  48849    0    0    0    0      212    0    0    0     0    0
eth0:1:  10894    0    0    0    0      210    0    0    0     0    0
eth0:2: 481325    0    0    0    0      259    0    0    0     0    0
eth0:3:  29178    0    0    0    0      215    0    0    0     0    0
```

You can see that the device files for an aliased ethernet device uses the format ethX:Y where X is the number for the ethernet card and Y is the number of the aliased device. Since aliased devices use the same ethernet card they must use the same network, after all you can't connect a single ethernet card to two networks.

Ethernet

The following provides some very brief background information on ethernet which will be useful in the rest of the chapter. Refer to the Ethernet HOWTO for more information.

Ethernet addresses

Every ethernet card has built into it a 48 bit address (called an Ethernet address or a Media Access Control (MAC) address). The high 24 bits of the address are used to assign a unique number to manufacturers of ethernet addresses and the low 24 bits are assigned to individual ethernet cards made by the manufacturer.

Some example ethernet addresses, you will notice that ethernet addresses are written using 6 tuples of HEX numbers, are listed below

```
00:00:0C:03:79:2F
00:40:F6:60:4D:A4
00:20:AF:A4:55:87
00:20:AF:A4:55:7B
```

Notice that the last two ethernet cards were made by the same manufacturer (with the manufacturers number of `00:20:AF`).

Ethernet is a broadcast medium

Every packet, often called an ethernet frame, of information sent on ethernet contains a source and destination MAC address. The packet is placed on a ethernet network and every machine, actually the ethernet card, on the network looks at the packet. If the card recognises the destination MAC as its own it "grabs" the packet and passes it to the Network access layer.

It is possible to configure your ethernet card so that it grabs all packets sent on the network. This is how it is possible to "listen in" on other people on a ethernet network.

A single ethernet network cannot cover much more than a couple of hundred meters. How far depends on the type of cabling used.

Converting hardware addresses to Internet addresses

The network access layer, the lowest level of the TCP/IP protocol stack is responsible for converting Internet addresses into hardware addresses, such as MAC addresses. This is how TCP/IP can be used over a large number of different networking hardware. As you might have guessed different networking hardware uses different addressing schemes.

Address Resolution Protocol

The mapping of ethernet addresses into Internet addresses is performed by the Address Resolution Protocol (ARPARP). ARP maintains a table that contains the translation between IP address and ethernet address.

When the machine wants to send data to a computer on the local ethernet network the ARP software is asked if it knows about the IP address of the machine (remember the software deals in IP addresses). If the ARP table contains the IP address the ethernet address is returned.

If the IP address is not known a packet is broadcast to every host on the local network, the packet contains the required IP address. Every host on the network examines the packet. If the receiving host recognises the IP address as its own, it will send a reply back that contains its ethernet address. This response is then placed into the ARP table of the original machine (so it knows it next time).

The ARP table will only contain ethernet addresses for machines on the local network. Delivery of information to machines not on the local network requires the intervention of routing software which is introduced later in the chapter.

arp

On a UNIX machine you can view the contents of the ARP table using the `arp` command. `arp -a` will display the entire table.

The following example shows how the arp cache for a computer is built as it goes. In the first use of the arp command you can see three machines in the cache, centaurus, draal and a ?. The ? is almost certainly one of the NT computers in the student labs at CQU. Draal is one of the Linux computers used by project students and centaurus is the gateway between the 138.77.37 network and the rest of the world.

```
[root@cq-pan logs]# /sbin/arp –a
centaurus.cqu.EDU.AU (138.77.37.1) at AA:00:04:00:0B:1C [ether] on eth0
draal.cqu.EDU.AU (138.77.37.100) at 00:20:AF:33:B5:BE [ether] on eth0
? (138.77.37.46) at <incomplete> on eth0
```

To see how new entries are added to the cache the next example shows the ping command. `ping` is often used to test a network connection and to see if a particular machine is alive. In this case I'm pinging pug, who also happens to be on the 138.77.37 network.

```
[root@cq-pan logs]# ping pug
PING pug.cqu.edu.au (138.77.37.102): 56 data bytes
64 bytes from 138.77.37.102: icmp_seq=0 ttl=64 time=19.0 ms

--- pug.cqu.edu.au ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 19.0/19.0/19.0 ms
```

Since we've now contacted pug and pug is on the same network as this machine its entry should now appear in the arp cache.

```
[root@cq-pan logs]# /sbin/arp –a
centaurus.cqu.EDU.AU (138.77.37.1) at AA:00:04:00:0B:1C [ether] on eth0
draal.cqu.EDU.AU (138.77.37.100) at 00:20:AF:33:B5:BE [ether] on eth0
pug.cqu.EDU.AU (138.77.37.102) at 00:20:AF:A4:3B:0F [ether] on eth0
? (138.77.37.46) at <incomplete> on eth0
```

There (s)he blows. If pug was not on the same local area network its ethernet address would not be added to the arp cache. Remember, ethernet addresses are only used to communicate with machines on the same ethernet network. For example, if I ping the machine www.cqu.edu.au it won't be added to the arp cache since it is on a different network.

```
[root@cq-pan logs]# ping www
PING plato.cqu.edu.au (138.77.5.4): 56 data bytes
64 bytes from 138.77.5.4: icmp_seq=0 ttl=63 time=1.7 ms

--- plato.cqu.edu.au ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.7/1.7/1.7 ms
```

SLIP, PPP and point to point

SLIP and PPP, used to connect machines via serial lines (and modems) are not broadcast media. They are simple "point-to-point" connections between two computers. This means that when information is placed on a SLIP/PPP connection only the two computers at either end of that connection can see the

information. SLIP/PPP are usually used when a computer is connected to a network via a modem or a serial connection.

This chapter does not provide any more discussion of SLIP/PPP. However all the basic concepts and the fundamental process for connecting a machine to the network are the same for SLIP/PPP as they are for ethernet. This is one of the advantages of TCP/IP networking being layered. Above a certain level, i.e. when the network interface is configured, the system works the same regardless of the hardware. Refer to the appropriate HOWTOs for more information.

Kernel support for networking

Ensuring that the kernel includes support for your networking hardware is only the first step. In order to supply certain network services it is necessary for them to be compiled into the kernel. The following is a list of some of the services that the Linux kernel can support. The list itself may be a touch out of date. If in doubt refer to the appropriate networking and kernel HOWTOs.

- IP accounting
  IP accounting must be compiled into the kernel and is configured with the `ipfwadm` command. IP accounting allows you to track the number of bytes and packets transmitted over the network connection. This is useful in situations where you must track the network usage of your users. For example, if you are a Internet Service Provider.

- IP aliasing
  Essentially, IP aliasing allows your computer to pretend it is more than one computer. In a normal configuration each network device is allocated a single IP address. However there are times when you wish to allocate multiple IP addresses to a computer with a single network interface. The most common example of this is web sites, for example, the websites `http://cq-pan.cqu.edu.au/`, `http://webclass.cqu.edu.au/`, and `http://webfuse.cqu.edu.au/` are all hosted by one computer. This computer only has one ethernet card and uses IP aliasing to create aliases for the ethernet card. The ethernet card's real IP address is 138.77.37.37 and its three alias addresses are 138.77.37.36, 138.77.37.59 and 138.77.37.108.

  Normally the interface would only grab the network packets addressed to 138.77.37.37 but with network aliasing it will grab the packets for all three addresses.

  You can see this in action by using the arp command. Have a look at the hardware addresses for the computers cq-pan, webclass and webfuse. What can you tell?

```
[david@draal david]$ /sbin/arp
Address                  HWtype  HWaddress           Flags Mask            Iface
centaurus.cqu.EDU.AU     ether   AA:00:04:00:0B:1C   C                      eth0
webfuse.cqu.EDU.AU       ether   00:60:97:3A:AA:85   C                        eth0
cq-pan.cqu.EDU.AU        ether   00:60:97:3A:AA:85   C                        eth0
science.cqu.EDU.AU       ether   00:00:F8:01:9E:DA   C                        eth0
```

```
borric.cqu.EDU.AU     ether   00:20:AF:A4:39:39   C                     eth0
webclass.cqu.EDU.AU   ether   00:60:97:3A:AA:85   C                     eth0
138.77.37.46                  (incomplete)                              eth0
```

- IP firewall
  This option allows you to use a Linux computer to implement a firewall. A firewall works by allowing you to selectively ignore certain types of network connections. By doing this you can restrict what access there is to your computer (or the network behind it) and as a result help increase security.

  The firewall option is closely related to IP accounting, for example it is configured with the same command, `ipfwadm`.

  Firewall support has changed in the newer 2.2 kernels. Please refer to the appropriate HOWTOs.
- IP encapsulation
  IP encapsulation is where the IP packet from your machine is wrapped inside another IP packet. This is of particular use mobile IP, IP multicast and the new buzzword Virtual Private Networks (VPNs).
- IPX
  IPX protocol is used in Novel Netware systems. Including IPX support in the Linux kernel allows a Linux computer to communicate with Netware machines.
- IPv6
  IPv6, version 6 of the IP protocol, is the next generation of which is slowly being adopted. IPv6 includes support for the current IP protocol. Linux support for IPv6 is slowly developing. You can find more information at http://www.v6.linux.or.jp/
- IP masquerade
  IP masquerade allows multiple computers to use a single IP address. One situation where this can be useful is when you have a single dialup connection to the Internet via an Internet Service Provider (ISP). Normally,such a dialup connection can only be used by the machine which is connected. Even if the dialup machine is on a LAN with other machines connected they cannot access the Internet. However with IP masquerading it is possible to allow all the machines on that LAN access the Internet.
- Network Address Translation
  Support for network address translation for Linux is still at an apha stage. Network address translation is the "next version" of IP masquerade. See `http://linas.org/linux/load.html` for more information.
- Mobile IP
  Since an IP address consists of both a network address and a host address it can normally only be used when a machine is connected to the network specified by the network address. Mobile IP allows a machine to be moved to other networks but still retain the same IP. IP encapsulation is used to send packets destined for the mobile machine to its new location. See for

http://www.hpl.hp.com/personal/Jean_Tourrilhes/MobileIP/mip.html more information.

- IP multicast
  IP multicast is used to send packets simultaneously to computers and separate IP networks. It is used for a variety of audio and video transmission. See http://www.teksouth.com/linux/multicast/ for more information.
- EQL
  EQL allows you to treat multiple point-to-point connections (SLIP, PPP) as a single logical TCP/IP connection.

By default the Linux kernel will have most of the networking services you require already compiled into it. However, if you want to make use of some of the additional features you need to check and possibly recompile the kernel with support for the required feature.

Configuring the connection

Having reached this stage it is assumed that you have connected (or inserted) your networking hardware (in)to your computer and have (if necessary) recompiled the kernel to provide the necessary networking support. This section provides an overview of configuring a network connection for a fairly typical local area network using ethernet. The steps are much the same for other types of hardare.

The Configuration Process

The configuration process includes the following steps

- Configure the devices
  Done either at system startup time (ethernet and other permanent connections) or by a user program (on-demand connections such as PPP over modems) this process configures the network devices with the appropriate information including IP address, network address etc.
- Configure the name resolver
  This step sets up the DNS so that your system can translate IP addresses into hostnames and vice versa
- Configure routing
  Informs the system how it is meant to send information from one network to another.

The following discusses how these steps are performed.

If you were planning to include some of the more advanced features available in the Linux kernel, such as IP masquerading, IP aliasing or EQL, you would have to perform additional steps as outlined in the appropriate HOWTOs or manual pages.

Configuration Related Tools and Files

Configuring the network interface generally makes use of the following tools/files

- command line tools such as `ifconfig`
  The standard tools used to perform the configuration you might use this from the command line or it might be performed by systems startup scripts.
- device configuration files
  Most distributions of Linux use text-based files to store the configuration information used by `ifconfig`. RedHat uses the `/etc/sysconfig/network` file and the `/etc/sysconfig/network-scripts` directory to contain these files which are used as the system starts up. The detail in these files includes IP address, netmask and broadcast addresses for the various devices.
- Network configuration files
  These files provide details for other network services such as DNS and routing.
- Startup files
  Chances are you will want your network to be automatically configured whenever the computer is turned on. This is where the system startup files enter the picture.
- GUI configuration tools
  To help ease the load most systems provide some sort of GUI tool which allows you to perform many of these tasks.

Configuring the device/interface

Earlier in the chapter the concept of a network device was introduced. The following section examines what is required to configure the network device so that it operates. Configuring the network device draws on some of the basic TCP/IP concepts introduced in previous sections.

The loopback device/interface

The loopback device is a special case. It is always present and is used to provide access to your own machine. Even if you do not have a network connection you will be able to use the loopback interface to test some of the basic networking services. The loopback interface always has the IP address `127.0.0.1`. Whenever you use the IP address `127.0.0.1` you are connecting to your own computer.

`ifconfig`

Network interfaces are configured using the `ifconfig` command and has the standard format for turning a device on

```
ifconfig device_name IP_address netmask netmask up
```

For example

- `ifconfig eth0 138.77.37.26 netmask 255.255.255.0 up`
  Configures the first ethernet address with the IP address of 138.77.37.26 and the netmask of 255.255.255.0.
- `ifconfig lo 127.0.0.1`
  Configures the loopback address appropriately.

Other parameters for the `ifconfig` command include

- `up` and `down`
  These parameters are used to take the device up and down (turn it on and off). ifconfig eth0 down will disable the eth0 interface and will require an ifconfig command like the first example above to turn it back on.
- `-arp`
  Will turn on/off the address resolution protocol for the specified interface.
- `-pointtopoint` *`addr`*
  Used to specify the IP address (*addr*) of the computer at the far end of a point to point link.

RedHat Configuration Files

Redhat uses the `/etc/sysconfig/network` file and the contents of the `/etc/sysconfig/network-scripts` directory to help in the configuration of network devices during the startup of the system. Hopefully you can draw some useful conclusions from the following examples

```
[root@cq-pan sysconfig]# cat network
NETWORKING=yes
FORWARD_IPV4=false
HOSTNAME=cq-pan.cqu.edu.au
DOMAINNAME=cqu.edu.au
GATEWAY=138.77.37.1
GATEWAYDEV=eth0
[root@cq-pan sysconfig]# cat  network-scripts/ifcfg-eth0
DEVICE=eth0
IPADDR=138.77.37.37
NETMASK=255.255.255.0
NETWORK=138.77.37.0
BROADCAST=138.77.37.255
ONBOOT=yes
```

The GUI configuration tools for RedHat Linux modify these files.

The script which actually starts networking on a RedHat Linux machine is `/etc/rc.d/init.d/network`

A more indepth explanation of these files please refer to the RedHat manuals for 6.1.

Configuring the name resolver

Once the device/interface is configured you can start using the network. However you'll only be able to use IP addresses. At this stage the networking system on your computer will not know how to resolve hostnames (convert hostnames into IP addresses). So if I was configuring a machine on the

138.77.37 subnet (this is the student subnet in the IT building) at CQU I would be able to execute commands like

```
telnet 138.77.37.37
```

but I would not be able to execute commands such as

```
telnet cq-pan.cqu.edu.au
```

Even though the IP address for the machine cq-pan.cqu.edu.au is 138.77.37.37 the networking on my machine doesn't know how to do the translation.

This is where the name resolver and its associated configuration files enter the picture. In particular the three files we'll be looking at are

- `/etc/resolv.conf`
  Specifies where the main domain name server is located for your machine.
- `/etc/hosts.conf`
  Allows you to specify how the name resolver will operate. For example, will it ask the domain name server first or look at a local file.
- `/etc/hosts`
  A local file which specifies the IP/hostname association between common or local computers.

The following is an excerpt from the NET-3 HOW-TO which describes these files in a bit more detail.

`/etc/resolv.conf`

The `/etc/resolv.conf` is the main configuration file for the name resolver code. Its format is quite simple. It is a text file with one keyword per line. There are three keywords typically used, they are:

- domain
  This keyword specifies the local domain name.
- search
  This keyword specifies a list of alternate domain names to search for a hostname
- nameserver
  This keyword, which may be used many times, specifies an IP address of a domain name server to query when resolving names

An example `/etc/resolv.conf` might look something like:

```
domain maths.wu.edu.au
search maths.wu.edu.au wu.edu.au
nameserver 192.168.10.1
nameserver 192.168.12.1
```

This example specifies that the default domain name to append to unqualified names (ie hostnames supplied without a domain) is maths.wu.edu.au and that if the host is not found in that domain to also try the wu.edu.au domain directly. Two nameservers entry are supplied, each of which may be called upon by the name resolver code to resolve the name.

/etc/host.conf

The /etc/host.conf file is where you configure some items that govern the behaviour of the name resolver code.

The format of this file is described in detail in the 'resolv+' man page. In nearly all circumstances the following example will work for you:

```
order hosts,bind
multi on
```

This configuration tells the name resolver to check the /etc/hosts file before attempting to query a nameserver and to return all valid addresses for a host found in the /etc/hosts file instead of just the first.

/etc/hosts

The /etc/hosts file is where you put the name and IP address of local hosts. If you place a host in this file then you do not need to query the domain name server to get its IP Address. The disadvantage of doing this is that you must keep this file up to date yourself if the IP address for that host changes. In a well managed system the only hostnames that usually appear in this file are an entry for the loopback interface and the local hosts name.

```
# /etc/hosts
127.0.0.1       localhost loopback
192.168.0.1     this.host.name
```

You may specify more than one host name per line as demonstrated by the first entry, which is a standard entry for the loopback interface.

Configuring routing

Having performed each of the preceding steps the networking on your computer will still not be working 100% correctly. For example, assume I'm adding a machine to the 138.77.37 subnet at CQU with the IP address as 138.77.37.105 and the hostname `fred`. I've configured the network interface and set up the following files

(For the following discussion it is important to realise that CQU has a class B address, 138.77, and creates subnets which look like class C address, i.e. 138.77.37, 138.77.1 and 138.77.5 are all separate subnets)

- `/etc/resolv.conf`

```
search cqu.edu.au
nameserver 138.77.5.6
nameserver 138.77.1.23
```

- `/etc/host.conf`

```
order hosts,bind
multi on
```

- `/etc/hosts`

```
127.0.0.1   localhost   localhost.localdomain
138.77.37.105     fred  fred.cqu.edu.au
138.77.37.37    cq-pan   cq-pan.cqu.edu.au
```

Now, see what happens when I execute the following commands

```
[david@fred david]$ ping cq-pan.cqu.edu.au
PING cq-pan.cqu.edu.au (138.77.37.37): 56 data bytes
64 bytes from 138.77.37.37: icmp_seq=0 ttl=63 time=1.1 ms
64 bytes from 138.77.37.37: icmp_seq=1 ttl=63 time=1.0 ms
64 bytes from 138.77.37.37: icmp_seq=2 ttl=63 time=1.0 ms

--- cq-pan.cqu.edu.au ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.0/1.0/1.1 ms

[root@fred network-scripts]# ping jasper.cqu.edu.au
ping: unknown host jasper.cqu.edu.au
```

Why the difference? We've setup the name resolution configuration files properly so why can't it resolve the name `jasper.cqu.edu.au` to the IP address `138.77.1.1`? Have a look at the IP addresses of the domain name servers specified in the `/etc/resolv.conf` file above? What can you tell about these hosts?

The major difference between the domain name servers and our new host `fred` is that they are on separate subnets. At this stage our host has not been told how it is meant to send information from its own subnet to other subnets (remember the discussion earlier in the chapter about arp and ethernet being a broadcast medium?).

`fred.cqu.edu.au` is able to use the `cq-pan.cqu.edu.au` hostname because it is specified in the `/etc/hosts` file and it can send information to that machine because it is on the same subnet. Because the domain name servers are on another subnet the networking software on the machine doesn't know how to communicate with them. An example of what happens can be seen in the following command where rather than use `jasper.cqu.edu.au`'s hostname we use the IP address.

```
[david@fred david]$ ping 138.77.1.1
PING 138.77.1.1 (138.77.1.1): 56 data bytes
ping: sendto: Network is unreachable
ping: wrote 138.77.1.1 64 chars, ret=-1
ping: sendto: Network is unreachable
ping: wrote 138.77.1.1 64 chars, ret=-1

--- 138.77.1.1 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

The solution to this problem is to configuring the routing software on our computer. Routing is the art of deciding how to send IP packets from one host to another, particularly where there are possibly multiple paths that could be used. In our example above we have to specify how the networking software is to deliver IP packets from our current subnet, 138.77.37, to other subnets.

Routing is a huge and complex topic. It is not possible to provide a detailed introduction in the confines of this text. If you need more information you should take a look at the relevant HOWTOs and especially the Linux Network Administrators Guide.

An explanation of routing tables and commands

The following is an excerpt rom the NET-3 HOW-TO which briefly describes the routing table and the commands used to manipulate it.

Ok, so how does routing work ? Each host keeps a special list of routing rules, called a routing table. This table contains rows which typically contain at least three fields, the first is a destination address, the second is the name of the interface to which the datagram is to be routed and the third is optionally the IP address of another machine

which will carry the datagram on its next step through the network. In linux you can see this table by using the following command:

```
# cat /proc/net/route
```

or by using either of the following commands:

```
# /sbin/route –n
# /bin/netstat -r
```

The routing process is fairly simple: an incoming datagram is received, the destination address (who it is for) is examined and compared with each entry in the table. The entry that best matches that address is selected and the datagram is forwarded to the specified interface. If the gateway field is filled then the datagram is forwarded to that host via the specified interface, otherwise the destination address is assumed to be on the network supported by the interface.

To manipulate this table a special command is used. This command takes command line arguments and converts them into kernel system calls that request the kernel to add, delete or modify entries in the routing table. The command is called '`route`'.

A simple example. Imagine you have an ethernet network. You've been told it is a class-C network with an address of 192.168.1.0. You've been supplied with an IP address of 192.168.1.10 for your use and have been told that 192.168.1.1 is a router connected to the Internet.

The first step is to configure the interface as described earlier. You would use a command like:

```
# ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up
```

You now need to add an entry into the routing table to tell the kernel that datagrams for all hosts with addresses that match 192.168.1.* should be sent to the ethernet device. You would use a command similar to:

```
# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
```

Note the use of the '-net' argument to tell the route program that this entry is a network route. Your other choice here is a '-host' route which is a route that is specific to one IP address.

This route will enable you to establish IP connections with all of the hosts on your ethernet segment. But what about all of the IP hosts that aren't on your ethernet segment ?

It would be a very difficult job to have to add routes to every possible destination network, so there is a special trick that is used to simplify this task. The trick is called the 'default' route. The default route matches every possible destination, but poorly, so that if any other entry exists that matches the required address it will be used instead of the default route. The idea of the default route is simply to enable you to say "and everything else should go here". In the example I've contrived you would use an entry like:

```
# route add default gw 192.168.1.1 eth0
```

The 'gw' argument tells the route command that the next argument is the IP address, or name, of a gateway or router machine which all datagrams matching this entry should be directed to for further routing.

So, your complete configuration would look like:

```
# ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up
# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
# route add default gw 192.168.1.1 eth0
```

These steps are actually performed automatically by the startup files on a properly configured Linux box.

Network "management" tools

You might ask, "Why the hell are we playing with all these text files and commands? Why can't we just use the nice GUI tools that come with RedHat". The simple answer is that knowing how to use a GUI tool isn't all that difficult, anyone can learn that. What's important for a computing professional, like a Systems Administrator, to know is what is going on underneath. There will be times when the GUI doesn't work or the problem you have can't be solved with the GUI. It is at times like this that you will need to understand what is going on underneath.

Having said that it can be a lot quicker to perform simple tasks using a GUI than with text files and a command line (depending on your personal preference). The following section introduces the GUI tools RedHat provides to manage and configure networking and also looks at a couple of other useful commands UNIX provides.

RedHat GUI Networking Tools

RedHat supplies a number of GUI administration tools which are all launched from the control-panel application by typing `control-panel` from a shell (you must be running X-Windows as control-panel is an X application). Each of the icons in the control panel window correspond to one of the GUI tools. Holding the mouse over the icon will cause it to display the name of the tool.

Of particular interest to this chapter is the network configuration tool which allows you to configure the hosts, name servers, devices and routing for your system.

You might also be interested in using linuxconf a popular GUI configuration tool which provides a number of interfaces.

nslookup

The nslookup command is used to query a name server and is supplied as a debugging tool. It is generally used to determine if the name server is working correctly and for querying information from remote servers.

nslookup can be used from either the command line or interactively. Giving nslookup a hostname will result in it asking the current domain name server for the IP address of that machine.

nslookup also has an ls command that can be used to view the entire records of the current domain name server.

For example

```
 [david@cq-pan:~]$ nslookup
Default Server:  circus.cqu.edu.au
Address:  138.77.5.6

> jasper
Server:  circus.cqu.edu.au
Address:  138.77.5.6

Name:    jasper.cqu.edu.au
Address:  138.77.1.1

> exit
[david@cq-pan:~]$ nslookup jasper
Server:  circus.cqu.edu.au
Address:  138.77.5.6

Name:    jasper.cqu.edu.au
Address:  138.77.1.1
```

netstat

The netstat command is used to display the status of network connections to a UNIX machine. One of the functions it can be used for is to display the contents of the kernel routing table by using the -r switch.

For example

The following examples are from two machines on CQU's Rockhampton campus. The first one is from telnet jasper

```
[david@cq-pan:~]$ netstat -rn
Kernel routing table
Destination     Gateway         Genmask         Flags Metric Ref Use    Iface
138.77.37.0     0.0.0.0         255.255.255.0   U     0      0   109130 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0     9206 lo
0.0.0.0         138.77.37.1     0.0.0.0         UG    0      0   2546951 eth0
bash$ netstat -rn
Routing tables
Destination          Gateway              Flags    Refcnt Use        Interface
127.0.0.1            127.0.0.1            UH       56     7804440    lo0
default              138.77.1.11          UG       23     1595585    ln0
138.77.32            138.77.1.11          UG       0      19621      ln0
138.77.16            138.77.1.11          UG       0      555        ln0
138.77.8             138.77.1.11          UG       0      385345     ln0
138.77.80            138.77.1.11          UG       0      0          ln0
```

```
138.77.72           138.77.1.11         UG       0      0          ln0
138.77.64           138.77.1.11         UG       0      0          ln0
138.77.41           138.77.1.11         UG       0      0          ln0
```

traceroute

For some reason or another, users on one machine cannot connect to another machine or if they can any information transfer between the two machines is either slow or plagued by errors. What do you do?

Remember it is not only the machines at the two ends you have to check. If the two machines are on different networks the information will flow through a number of gateways and routers. It might be one of the gateway machines that is causing the problem.

The `traceroute` command provides a way of discovering the path taken by information as it goes from one machine to another and can be used to identify where problems might be occurring. On the Internet that path may not always be the same.

For example

The following are the results of a number of executions of traceroute from the machine `aldur` (`138.77.36.29`).

In the first example the machine `knuth` is on the same network as `aldur`. This means that the information can get their directly.

```
 bash$ traceroute knuth
traceroute to knuth.cqu.edu.au (138.77.36.20), 30 hops max, 40 byte
packets
1 knuth.cqu.EDU.AU (138.77.36.20) 2 ms 2 ms 2 ms
```

`jasper` is one network away from `aldur`

```
 bash$ traceroute jasper
traceroute to jasper.cqu.edu.au (138.77.1.1), 30 hops max, 40 byte
packets
1 centaurus.cqu.EDU.AU (138.77.36.1) 1 ms 1 ms 1 ms
2 jasper.cqu.EDU.AU (138.77.1.1) 2 ms 1 ms 1 ms
```

A machine still on the CQU site but a little further away

```
 bash$ traceroute jade
traceroute to jade.cqu.edu.au (138.77.7.2), 30 hops max, 40 byte
packets
1 centaurus.cqu.EDU.AU (138.77.36.1) 1 ms 1 ms 1 ms
2 hercules.cqu.EDU.AU (138.77.5.3) 4 ms 2 ms 12 ms
3 jade.cqu.EDU.AU (138.77.7.2) 3 ms 13 ms 3 ms
```

A host still in Australia (but a long way from CQU)

```
 bash$ traceroute archie.au
traceroute to archie.au (139.130.23.2), 30 hops max, 40 byte packets
1 centaurus.cqu.EDU.AU (138.77.36.1) 1 ms 1 ms 1 ms
2 tucana.cqu.EDU.AU (138.77.5.27) 2 ms 2 ms 2 ms
3 138.77.32.10 (138.77.32.10) 5 ms 5 ms 5 ms
4 qld.gw.au (139.130.60.1) 21 ms 13 ms 51 ms
5 national.gw.au (139.130.48.1) 35 ms 36 ms 40 ms
6 plaza.aarnet.edu.au (139.130.23.2) 38 ms 35 ms 68 ms
```

A host in the Eastern United States.

```
bash$ traceroute sunsite.unc.edu
traceroute to knuth.cqu.edu.au (139.130.23.2), 30 hops max, 40 byte packets
1 centaurus.cqu.EDU.AU (138.77.36.1) 1 ms 1 ms 1 ms
2 tucana.cqu.EDU.AU (138.77.5.27) 2 ms 2 ms 3 ms
3 138.77.32.10 (138.77.32.10) 5 ms 5 ms 5 ms
4 qld.gw.au (139.130.60.1) 13 ms 20 ms 13 ms
5 national.gw.au (139.130.48.1) 51 ms 36 ms 36 ms
6 usa.gw.au (139.130.29.5) 37 ms 36 ms 38 ms
7 usa-au.gw.au (203.62.255.1) 233 ms 252 ms 264 ms
8 * * t3-0.enss144.t3.nsf.net (192.203.230.253) 224 ms
9 140.222.8.4 (140.222.8.4) 226 ms 236 ms 258 ms
10 t3-3.cnss25.Chicago.t3.ans.net (140.222.25.4) 272 ms 293 ms 266 ms
11 t3-0.cnss40.Cleveland.t3.ans.net (140.222.40.1) 328 ms 270 ms 300 ms
12 t3-1.cnss48.Hartford.t3.ans.net (140.222.48.2) 325 ms 355 ms 289 ms
13 t3-2.cnss32.New-York.t3.ans.net (140.222.32.3) 284 ms 319 ms 347 ms
14 t3-1.cnss56.Washington-DC.t3.ans.net (140.222.56.2) 352 ms 299 ms 305 ms
15 t3-1.cnss72.Greensboro.t3.ans.net (140.222.72.2) 319 ms 344 ms 310 ms
16 mf-0.cnss75.Greensboro.t3.ans.net (140.222.72.195) 343 ms 320 ms *
17 cnss76.Greensboro.t3.ans.net (192.103.68.6) 338 ms 319 ms 355 ms
18 192.103.68.50 (192.103.68.50) 338 ms 330 ms 330 ms
19 rtp5-gw.ncren.net (128.109.135.254) 357 ms 361 ms *
20 * rtp2-gw.ncren.net (128.109.70.253) 359 ms 334 ms
21 128.109.13.2 (128.109.13.2) 374 ms 411 ms 451 ms
22 * calypso-2.oit.unc.edu (198.86.40.81) 418 ms 415 ms
```

There are now a number of visual versions of traceroute, http://www.visualroute.com/, is one of them

Exercises

15.3. In the above example examine the times between machines 6 & 7. Why do you think it takes so long to get from machine 6 to machine 7?

Conclusions

Network protocols are known for consisting of a number of layers. Connecting a Linux box to a network also has a number of layers

- Select the appropriate hardware
- Compile an appropriate kernel
- Configure the network devices with appropriate settings
- Configure the DNS and routing services
- Configure any additional networking services

As with network protocol layers, the layers in setting up a network connection also hide detail. For example, once you have connected the hardware and recompiled the kernel configuring network devices is very similar regardless of the networking hardware being used.

Review Questions

15.1

"The Net", a movie with Sandra Bullock, contained images of a few screens with what appeared to be IP addresses. Some of those supposed IP addresses are listed below. Are they IP addresses? If not, why not?

1. 128.234.15
2. 23.75.345.200
3. 75.258.34.164

15.2

What UNIX commands would you use for the following tasks

1. checking a domain name server for the IP address of the machine `www.seven.com.au`
2. finding out whether or not your computer can access, via the network, another machine,
3. finding out what machines information passes through as it goes from your machine to `www.whitehouse.gov`
4. configure a network interface,
5. display the routing table of your UNIX machine,
6. display the ethernet address of your UNIX machine.

15.3

Explain the relevance of each of the following

1. `/etc/hosts`
2. `/etc/resolv.conf`
3. `/etc/networks`
4. `/etc/rc.d/rc.inet1`
5. a gateway

15.4

It has been suggested that the “layering” of the network configuration steps means that configuring the network devices for a PPP connection will be similar to that for an ethernet connection. Refer to the appropriate manual pages and HOWTOs and compare the steps involved in making a PPP connection. For example

- Where is the device configuration information stored?
- How is the network device configured?
- Are the any similarities or differences with network configuratio?

- What about DNS and routing configuration, is there any similarities?

15.5

You've just started administering a new Linux computer and executed the following three `ifconfig` commands to discover information about the configuration of the network devices on this machine.

Using this output answer the following questions (HINT: remember the discussion of IP aliasing earlier in this chapter?)

1. List the network and host portions of the IP address for each of the network devices listed in the output of these commands.

What does the output of these commands tell you about the network configuration of these machines?

What would the `/proc/net/dev` file for this system look like?

Can you see what is wrong with the configuration of the networking of this system?

```
[root@cq-pan logs]# /sbin/ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:60:97:3A:AA:85
inet addr:138.77.37.37 Bcast:138.77.37.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:61183404 errors:59 dropped:59 overruns:0
TX packets:77722967 errors:0 dropped:0 overruns:0
Interrupt:9 Base address:0xff00

[root@cq-pan logs]# /sbin/ifconfig eth0:1
eth0:1 Link encap:Ethernet HWaddr 00:60:97:3A:AA:85
inet addr:138.77.37.59 Bcast:138.77.37.255 Mask:255.255.255.0
UP BROADCAST RUNNING MTU:1500 Metric:1
RX packets:10894 errors:0 dropped:0 overruns:0
TX packets:210 errors:0 dropped:0 overruns:0

[root@cq-pan logs]# /sbin/ifconfig eth0:2
eth0:2 Link encap:Ethernet HWaddr 00:60:97:3A:AA:85
inet addr:138.77.38.60 Bcast:138.77.38.255 Mask:255.255.255.0
UP BROADCAST RUNNING MTU:1500 Metric:1
RX packets:481325 errors:0 dropped:0 overruns:0
TX packets:259 errors:0 dropped:0 overruns:0
```

Chapter 16

Network Applications

Introduction

The previous chapter looked at how you connect a Linux box to a network and provide some basic services. That is not enough information to produce a useful Linux machine. You need to know how to configure and manage the higher level network services which are expected today including print/file sharing, electronic mail, File Transfer Protocol, World-Wide Web and others.

That's where this chapter comes in. It aims to provide an overview of how network applications work,how they operate and how they are configured. There is no way a single chapter can provide this information about all the available network applications. There are hundreds of them and each one can be quite complex. Instead this chapter focuses on the fundamentals, the concepts which are common to all these applications. If you are comfortable with this knowledge then learning how to configure a new application is quite simple. The chapter closes with a detailed look at some specific network services including file/print sharing, messaging (email) and the World-Wide Web.

Other Resources

Other available resources which examine similar material include

- HOW-TOs
  Firewall, IPCHAINS, Intranet Server (though it is a little dated), Mail, Mail User, NFS, NIS, Networking Overview (gives a very good overview of topics related to both networking chapters), SMB, VPN, Virtual Services, WWW, Apache SSL PHP/FI
- Mini HOW-TOs
  Apache SSL PHP/FI, Automount, Cipe+Masquerading, ISP Connectivity, NFS-Root, NFS-Root-Client, Qmail+MH, Remove Boot, Remote X Apps, Sendmail Address Rewrite, Sendmail+UUCP, Secure POP via SSH.
- LAME
  Sections on DNS Configuration, sections on Windows and Mac file and print sharing, NFS section, configuring the Apache Web server, configuring the Squid HTTP caching proxy, Configuring sendmail.
- Apache website http://www.apache.org/
- Samba website http://www.samba.org
- The RedHat reference and getting started guide has additional information about many of these topics.

An expanded an up to date list of resources can be found on the 85321 website.

How it all works

So what is the common details about all the network applications. How do they work? This section provides a general answer to these questions.

The provision of network services like FTP, telnet, e-mail and others relies on these following components

- network ports,
  Network ports are the logical (that means that ports are an imaginary construct which exists only in software) connections through which the information flows into and out of a machine. A single machine can have thousands of programs all sending and receiving information via the network at the same time. The delivery of this information to the right programs is achieved through ports. Generally each program must have its own port.
- network daemons,
  Network daemons are the programs running on the network server machines that sit listening at pre-defined ports waiting for connections from other hosts. These daemons wait for a request, perform some action and send a response back to the program that requested the action. The program which requested the action is a network client.
- network clients, and
  Users access network services using client programs. Example network client programs include Netscape, Eudora and the `ftp` command on a UNIX machine. The client programs turn user requests (e.g. typing in the URL http://www.linux.org/) into a request which is sent to a network daemon. The requests and responses which flow between network daemons and network clients must take part in some agreed upon format, a network protocol.
- network protocols.
  Network protocols specify how the network clients and servers communicate. They define the small "language" which both use for communication.

The following sections of this chapter go into more detail about each of these sections.

Ports

All network protocols, including `http ftp SMTP`, use either TCP or UDP to deliver information. TCP and UDP are referred to as transport protocols. Each transport protocol has its own characteristics and which one is used depends on the type of communication which occurs.

However, one thing is common between both transport protocols. The addresses they use to identify the source (where they are coming from) and the destination (where they are going to). Obviously the first component of the

source/destination address is the IP address, this identifies the computer. The next component is the port number on that computer.. Every TCP or UDP header contains two 16 bit numbers that are used to identify the source port (the port through which the information was sent) and the destination port (the port through which the information must be delivered.) The IP address is stored in the IP header.

Since port numbers are 16 bit numbers, there can be approximately 64,000 (2^{16} is about 64,000) different ports. Some of these ports are used for predefined purposes. The ports 0-256 are used by the network servers for well known Internet services (e.g. telnet, FTP, SMTP). Ports in the range from 256-1024 are used for network services that were originally UNIX specific. Network client programs and other programs should use ports above 1024.

Table 16.1 lists some of the port numbers for well known services.

Port number	Purpose
20	`ftp-data`
21	`ftp`
23	`telnet`
25	`SMTP` (mail)
80	`http` (WWW)
119	`nntp` (network news)

Table 16.1
Reserved Ports

This means that when you look at a TCP/UDP packet and see that it is addressed to port 25 then you can be sure that it is part of an email message being sent to a SMTP server. A packet destined for port 80 is likely to be a request to a Web server.

Reserved ports

So how does the computer know which ports are reserved for special services? On a UNIX computer this is specified by the file `/etc/services`. Each line in the `services` file is of the format

```
service-name port/protocol aliases
```

Where `service-name` is the official name for the service, `port` is the port number that it listens on, `protocol` is the transport protocol it uses and `aliases` is a list of alternate names.

The following is an extract from an example `/etc/services` file. Most `/etc/services` files will be the same, or at least very similar.

```
echo 7/tcp
echo 7/udp
discard 9/tcp sink null
discard 9/udp sink null
systat 11/tcp users
daytime 13/tcp
daytime 13/udp
```

```
ftp-data 20/tcp
ftp 21/tcp
telnet 23/tcp
smtp 25/tcp mail
nntp 119/tcp usenet # Network News Transfer
ntp 123/tcp # Network Time Protocol
```

You should be able to match some of the entries in the above example, or in the `/etc/services` file on your computer, with the entries in Table 16.1.

Exercises

16.1. Examine your `/etc/services` file and discover the port on which the following protocols are used
`http`
`ssh`
`pop3`

Look at ports, `netstat`

The `netstat` command can be used for a number of purposes including looking at all of the current active network connections. The following is an example of the output that `netstat` can produce (it's been edited to reduce the size).

```
[david@cq-pan:~]$ netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        (State) User
root
tcp        1   7246 cq-pan.cqu.edu.au:www  lore.cs.purdue.e:42468 CLOSING root
tcp        0      0 cq-pan.cqu.edu.au:www  sdlab142.syd.cqu.:1449 CLOSE   root
tcp        0      0 cq-pan.cqu.edu.au:www  dialup102-4-9.swi:1498 FIN_WAIT2 root
tcp        0  22528 cq-pan.cqu.edu.au:www  205.216.78.103:3058    CLOSE   root
tcp        1  22528 cq-pan.cqu.edu.au:www  barney.poly.edu:47547  CLOSE   root
tcp        0      0 cq-pan.cqu.edu.au:www  eda.mdc.net:2395       CLOSE   root
tcp        0  22528 cq-pan.cqu.edu.au:www  eda.mdc.net:2397       CLOSE   root
tcp        0      0 cq-pan.cqu.edu.au:www  cphppp134.cyberne:1657 FIN_WAIT2 root
tcp        0  22528 cq-pan.cqu.edu.au:www  port3.southwind.c:1080 CLOSE   root
tcp        0      9 cq-pan.cqu.edu.:telnet dinbig.cqu.edu.au:1107 ESTABLISHED root
tcp        0      0 cq-pan.cqu.edu.au:ftp  ppp2-24.INRE.ASU.:1718 FIN_WAIT2 root
```

Explanation

Table 16.2 explains each column of the output. Taking the column descriptions from the table, it is possible to make some observations

- All of the entries, but the last two, are for people accessing this machine's (`cq-pan.cqu.edu.au`) World-Wide Web server.
  You can say this because of `cq-pan.cqu.edu.au:www`. This tells us that the port on the local machine is the www port (port 80).
- In the second last entry, I am telneting to `cq-pan` from my machine at home.
  At that stage my machine at home was called `dinbig.cqu.edu.au`.
  The telnet client is using port 1107 on `dinbig` to talk to the telnet daemon.
- the last entry is someone connecting to CQ-PAN's ftp server,

- the connection for the first entry is shut down but not all the data has been sent (this is what the `CLOSING` state means).
  This entry, from a machine from Purdue University in the United States, still has 7246 bytes still to be acknowledged

Column name	Explanation
`Proto`	the name of the transport protocol (TCP or UDP) being used
`Recv-Q`	the number of bytes not copied to the receiving process
`Send-Q`	the number of bytes not yet acknowledged by the remote host
`Local Address`	the local hostname (or IP address) and port of the connection
`Foreign Address`	the remote hostname (or IP address) and remote port
`State`	the state of the connection (only used for TCP because UDP doesn't establish a connection), the values are described in the man page
`User`	some systems display the user that owns the local program serving the connection

Table 16.2
Columns for `netstat`

Network daemons

The `/etc/services` file specifies which port a particular protocol will listen on. For example `SMTP` (Simple Mail Transfer Protocol, the protocol used to transfer mail between different machines on a TCP/IP network) uses port 25. This means that there should be a network daemon that listens for SMTP connections on port 25 and knows what to do with those connections.

This begs some questions

- How do we know which program acts as the network daemon for which protocol?
- How is that program started?

How network daemons start

There are two methods by which network daemons are started

- by startup scripts
  Daemons started in this manner will show up in a `ps` list of all the current running processes. These daemons are always running, waiting for a connection on the specified port. This means that the daemon is using up system resources (RAM etc) because it is always in existence but it also means that it is very quick to respond when requests arrive for their services.

- by the `inetd` daemon
  The `inetd` daemon listens at a number of ports and when information arrives, it starts the appropriate network daemon for that port. Which daemon, for which port, is specified in the configuration file `/etc/inetd.conf`.

Starting a network daemon via `inetd` is usually done when there aren't many connections for that daemon. If a network daemon is likely to get a large number of connections (a busy mail or WWW daemon for example) the daemon for that service should be started in the system startup files and always listen on the port.

The reason for this is overhead. Using `inetd` takes longer because for every connection it needs to first create a new process (and we've seen already that creating new processes can be a relatively expensive process). When the daemon is already running and listening to the port it simply starts handling the request.

Of course the draw back with starting daemons in the startup scripts is that they are always there consuming RAM and other resources. Even if they aren't being used.

inetd

The `/etc/inetd.conf` file specifies the network daemons that the `inetd` daemon should execute. The `inetd.conf` file consists of one line for each network service using the following format (Table 16.3 explains the purpose of each field).

```
service-name socket-type protocol flags user daemon_program args
```

Field	Purpose
`service-name`	The service name, the same as that listed in `/etc/services`
`socket-type`	The type of data delivery services used (we don't cover this). Values are generally `stream` for TCP, `dgram` for UDP and `raw` for direct IP
`protocol`	the transport protocol used, the name matches that in the `/etc/protocols` file
`flags`	how `inetd` is to behave with regards this service (not explained any further)
`user`	the username to run the daemon as, usually root but there are some exceptions, generally for security reasons
`daemon_program`	the full path to the program to run as the daemon
`args`	command line arguments to pass to the daemon program

Table 16.3
Fields of /etc/inetd.conf

The following is an excerpt from the /etc/inetd.conf on a RedHat Linux machine. Notice that some of the entries have been commented out. This

means that these services will not be operational. No daemon listening for connections means the service won't work.

```
shell   stream  tcp     nowait  root            /usr/sbin/tcpd  in.rshd
login   stream  tcp     nowait  root            /usr/sbin/tcpd  in.rlogind
#exec   stream  tcp     nowait  root            /usr/sbin/tcpd  in.rexecd
#comsat dgram   udp     wait    root            /usr/sbin/tcpd  in.comsat
talk    dgram   udp     wait    nobody.tty      /usr/sbin/tcpd  in.talkd
ntalk   dgram   udp     wait    nobody.tty      /usr/sbin/tcpd  in.ntalkd
```

How it works

Whenever the machine receives a request on a port (on which the `inetd` daemon is listening on), the `inetd` daemon decides which program to execute on the basis of the `/etc/inetd.conf` file.

Exercises

16.2. `top` is a UNIX command which will give you a progressive display of the current running processes. Use `top` to observer what happens when a network daemon is started. For example, start `top` and then try to `telnet` or `ftp` to your machine. Can you see the appropriate daemon start? (Remember you should be able to use the hostname localhost for your own machine even if you are not on a network.)

16.3. What happens if you change the `/etc/inetd.conf` file? Does the `inetd` daemon pick up the change automatically? How would you notify `inetd` of the change?
Note: you WILL have to experiment to find out the answer to this question. It isn't included in the study material. A suggested experiment is the following: try the command `telnet localhost`, this should cause `inetd` to do some work; if it works, comment out the entry in the `inetd.conf` file for the telnet service try the first command again. Does it work? If it does then `inetd` hasn't seen the change. How do you tell it?

Network clients

All of you will have used a number of network client programs. If you are reading this online you may well be using a WWW browser. It's a network client program. Checking your mail makes use of a network client. A network client is simply a program (whether it is text based or a GUI program) that knows how to connect to a network daemon, pass requests to the daemon and then receive replies.

The telnet client

By default when you use the command `telnet jasper`, the `telnet` client program will attempt to connect to port 23 of the host `jasper` (23 is the `telnet` port as listed in `/etc/services`).

It is possible to use the `telnet` client program to connect to other ports. For example the command `telnet jasper 25` will connect to port 25 of the machine `jasper`.

The usefulness and problem with this will be discussed on the next couple of pages.

Network protocols

Each network service generally uses its own network protocol that specifies the services it offers, how those services are requested and how they are supplied. For example, the `ftp` protocol defines the commands that can be used to move files from machine to machine. When you use a command line `ftp` client, the commands you use are part of the `ftp` protocol.

Request for comment (RFCs)

For protocols to be useful, both the client and daemon must agree on using the same protocol. If they talk different protocols then no communication can occur. The standards used on the Internet, including those for protocols, are commonly specified in documents called Request for Comments (RFCs). (Not all RFCs are standards). Someone proposing a new Internet standard will write and submit an RFC. The RFC will be distributed to the Internet community who will comment on it and may suggest changes. The standard proposed by the RFC will be adopted as a standard if the community is happy with it.

Protocol	RFC
FTP	959
Telnet	854
SMTP	821
DNS	1035
TCP	793
UDP	768

Table 16.4
RFCs for Protocols

Table 16.4 lists some of the RFC numbers which describe particular protocols. RFCs can and often are very technical and hard to understand unless you are familiar with the area (the RFC for `ftp` is about 80 pages long).

Exercises

16.4. Take a look at http://www.faqs.org/ the maintain a collection of FAQs from Usenet news and also provide access to the RFCs. Use this site to view the RFC for SMTP. Take a look through it to get an idea of what is there. The direct URL you want is http://www.faqs.org/rfcs/rfc821.html (at least at the time of writing).

Text based protocols

Some of these protocols `smtp ftp nntp http` are text based. They make use of simple text-based commands to perform their duty. Table 16.5 contains a list of the commands that `smtp` understands. `smtp` (simple mail transfer protocol) is used to transport mail messages across a TCP/IP network.

Command	Purpose
HELO *hostname*	startup and give your hostname
MAIL FROM: *sender-address*	mail is coming from this address
TO: *recipient-address*	please send it to this address
VRFY *address*	does this address actually exist (verify)
EXPN *address*	expand this address
DATA	I'm about to start giving you the body of the mail message
RSET	oops, reset the state and drop the current mail message
NOOP	do nothing
DEBUG [*level*]	set debugging level
HELP	give me some help please
QUIT	close this connection

Table 16.5
SMTP commands

How it works

When transferring a mail message a client (such as Eudora) will connect to the SMTP daemon (on port 25). The client will then carry out a conversation with the daemon using the commands from Table 16.5. Since these commands are just straight text you can use `telnet` to simulate the actions of an email client.

Doing this actually has some real use. I often use this ability to check on a mail address or to expand a mail alias. The following shows an example of how I might do this.

The text in bold is what I've typed in. The text in italics are comments I've added after the fact.

```
beldin:~$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220-beldin.cqu.edu.au Sendmail 8.6.12/8.6.9 ready at Wed, 1 May 1996
13:20:10 +1 000
220 ESMTP spoken here
vrfy david  check the address david
```

```
250 David Jones <david@beldin.cqu.edu.au
vrfy joe  check the address joe
550 joe... User unknown
vrfy postmaster  check the address postmaster
250 <postmaster@beldin.cqu.edu.au
expn postmaster  postmaster is usually an alias, who is it really??
250 root <postmaster@beldin.cqu.edu.au
```

Since 1996, when the above exercise was performed, the Internet has changed a lot. Someof the features shown above may not be supported by some mail servers due to concerns about security and mail spamming (where you are sent email you didn't ask for from people you don't know, usually trying to get you to give them money).

Mail spoofing

This same approach can be used to spoof mail, that is, send email as someone you are not. This is one of problems with Internet mail. The following is an example of how it's done.

```
bash$ telnet aldur 25  connect to the smtp port (see /etc/services)
Trying 138.77.36.29 ...
Connected to aldur.cqu.edu.au.
Escape character is '^]'.
220 aldur.cqu.edu.au Amix Smail3.1.28.1 #2 ready at Sun, 28 Aug 94
12:04 EST
helo aldur  tell the machine who I am (the name of another machine not a user)
250 aldur.cqu.edu.au Hello aldur
mail from: god@heaven.com  this is who the mail is coming from
250 <god@heaven> ... Sender Okay
data  I want to enter some data which is the message
503 Need RCPT (recipient)  can't do that yet, must tell it who to send message to
rcpt: david@aldur
500 Command unrecognized  oops, typed it wrong
rcpt to: david@aldur
250 <david@aldur> ... Recipient Okay
data
354 Enter mail, end with "." on a line by itself
You have been a naughty boy  type in the message
.
250 Mail accepted
quit  bye, bye
221 aldur.cqu.edu.au closing connection
Connection closed by foreign host.
```

There are methods which can be used to identify email sent in this way.

Exercises

16.5. Using the "`telnet`" approach connect to an `ftp` daemon and a `http` daemon. What commands do they recognise? You might want to refer to the RFCs for those protocols to find out.

Security

Putting your computer on a network, especially the Internet, makes it accessible to a lot of other people and not all of those people are nice. It is essential that you put in place some sort of security to protect your system from these nasty people. The next chapter takes a more indepth look at security. In this section we examine some of the steps you can take to increase the security of your system.

TCPWrappers/tcpd

The following are entries from two different `/etc/inetd.conf` files. Both are the entries dealing with the `telnet` service. The second entry is from a "modern" Linux machine, the first is from an earlier UNIX machine.

```
telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd
telnet stream tcp nowait root /usr/sbin/tcpd  /usr/sbin/in.telnetd
```

The difference

Do you notice the difference? The program being run on the Linux machine is `/usr/sbin/tcpd`. If you examine the entries in a Linux machine's `/etc/inetd.conf` you will find that this program is executed for **all** (almost) network services.

`tcpd` is the public domain program TCPWrappers that comes standard on all Linux machines. It is a special daemon that provides some additional services including added security, access control and logging facilities for all network connections. TCPWrappers works by being inserted between the `inetd` daemon and the various network daemons that are executed by `inetd`.

Figures 16.1 and 16.2 demonstrate the difference.

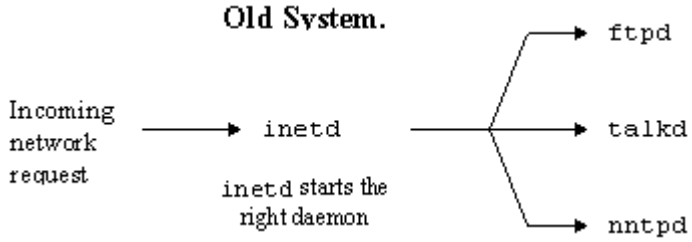


Figure 16.1
`inetd` by itself

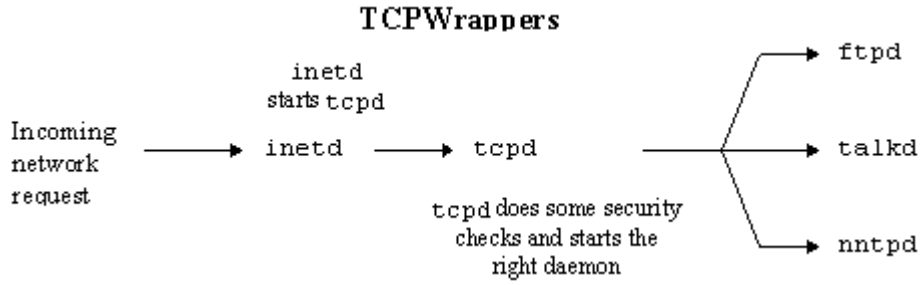


Figure 16.2
inetd with tcpd

tcpd features

`tcpd works as follows`

- a request for a particular network request is received,
- the configuration of `inetd` is such that `tcpd` is executed rather than the actual daemon for this request,
- `tcpd` logs the request via `syslog`,
  On RedHat each connection is logged into the file `/var/log/secure`. Information stored includes the time it was made, the host trying to make the connection and the name of the network service being requested. An example entry looks like

```
May  1 12:13:46 beldin in.telnetd[684]: connect from localhost
```

- `tcpd` then performs a number of checks,
  These checks make use of some the extra features of `tcpd` including
- pattern-based access control.
  This allows you to specify which hosts are allowed (or not) to use a particular network service. You can use this feature to restrict who can make use of your network services. tcpd also allows you to execute UNIX commands when a particular type of connection occurs.

Exercises

16.6. The manual page for `tcpd` says that more information about the access control features of `tcpd` can be found on the `hosts_access`(5) manual page. What command would you use to view this page?

- hostname verification,
  Some of the network protocols rely on hostnames for authentication. For example, you may only be able to use the `rsh` command if your computer is called `beldin.cqu.edu.au`. It is possible for people to setup computers that will pretend to be another hostname. `tcpd` offers a feature which will verify that a host is really who they say they are.
- protection against host address spoofing.
  It is also possible to spoof an IP address. That is, packets being sent from machine are modified to look as if they are being sent from another,

trusted, machine. `tcpd` offers a feature to detect and reject any connections of this type.

Exercises

16.7. Using `tcpd` how would you achieve the following
– Configure your machine so there are no network services available.
– Once you've done this attempt to `telnet` and `ftp` to your machine.
Keep this `tcpd` configuration for all the exercises in this group.

16.8. What effect would the previous question have on the ability for your machine to receive email?

16.9. Modify your `tcpd` configuration to allow the receipt of email.

16.10. Try connecting to the Web daemon on your machine. Assuming you have a standard RedHat installation you should still be able to connect to the Web daemon. Why can you still do this? Shouldn't your `tcpd` configuration have stopped this?

Other methods for securing a network connection are discussed in the security chapter.

What's an Intranet?

Intranets are the latest buzzword in the computer industry. The buzzword makers have finally realised the importance of the Internet (and the protocols with which it was constructed) and have started adopting it for a number of purposes. An intranet is basically a local area network used by an organisation that uses the Internet protocols to provide the services normally associated with a LAN plus offering Internet services (but not necessarily Internet access).

Services on an Intranet

The following is a list of the most common services that an Intranet might supply (by no means all of them). This is the list of services we'll discuss in more detail in this chapter. The list includes

- file sharing,
  The common ability to share access to applications and data files. It's much simpler to install one copy of an application on a network daemon than it is to install 35 copies on each individual PC.
- print sharing, and
  The ability for many different machines to share a printer. It is especially economically if the printer is an expensive, good quality printer.
- electronic mail.
  Sometimes called messaging. Electronic mail is fast becoming an essential tool for most businesses.
- Web serving

File and print sharing

There is a famous saying in the computing field.

> *The nice thing about standards is that there are so many to choose from.*

This statement is especially true in the area of sharing printers and files in a local area network. Some of the different protocols are outlined in Table 16.6 which also describes the origins of each protocol.

Name	Description
Server Message Block (SMB)	The protocol used by Windows for Workgroups, 95 and NT and OS/2 and a couple of others. Becoming the protocol with the largest number of clients.
Netware	Netware is the term used to describe Novell's network OS. Includes the protocols IPX and NCP (amongst others). A very popular, but possibly dying, network operating system (NOS).
Appletalk	The networking built-in to all Macintosh computers. Many Macs now use MacTCP which allows them to "talk" TCP/IP.
Network File System (NFS)	The traditional UNIX based file sharing system. NFS clients and daemons are available for most platforms.

Table 16.6
Protocols for sharing files and printers

The "native" form of file sharing on a UNIX machine is NFS. If you wanted to share files between UNIX machines, NFS would be the choice.

Due to a number of free software packages, Linux, and most versions of UNIX, can actually act as a server for all of the protocols listed above. Due to the popularity of the Windows family of operating systems, the following will examine the SMB protocols.

Samba

Samba is a piece of software, originally written by Andrew Tridgell (a resident of Canberra), and now maintained by a large number of people from throughout the world. Samba allows a UNIX machine to act as a file and print server for clients running Windows for Workgroups, Windows 95, NT and a couple of other operating systems.

The combination of Linux and Samba is possibly the cheapest way of obtaining a server for a Intranet.

The following is a very simple introduction to how you might use Samba on a RedHat machine. This process is much simpler on RedHat as Samba comes pre-configured. The readings down below provide much more information about Samba.

The configuration file for Samba is `/etc/smb.conf`. An entry in this configuration file which allows a user's home directory to be exported to SMB clients is the following

```
[homes]
   comment = Home Directories
   browseable = no
   read only = no
   preserve case = yes
   short preserve case = yes
   create mode = 0750
```

If your Linux machine happens to be on a network and you have a Win95/NT or even 3.11 machine on the same network, you should be able to connect to your home directory from that Windows machine using the standard approach for mapping a network drive. Figure 16.3 is the dialog box on a Windows 95 machine.

Figure 16.3
Dialog box for mapping a network drive.

In this example, the name of my Linux computer is `beldin` and my username on `beldin` is `david`. Once connected, I can now read and write files from my home directory from within Windows.

Chances are most of you will not have a local area network (LAN) at home that has your RedHat Linux machine and another Windows machine connected. This makes it difficult for you to recreate the above example. Luckily Samba comes with a program called `smbclient`. `smbclient` is a UNIX program which allows you to connect to Samba shares. This means when you use `smbclient` you are simulating what would happen if you were using a Windows machine. The following is an example of using `smbclient` to connect to the same share as in the Windows example above.

```
[david@beldin david]$ smbclient '\\beldin\david'
Added interface ip=138.77.36.28 bcast=138.77.36.255 nmask=255.255.255.0
Unknown socket option TCP_NODELAY
Server time is Fri Feb  6 14:04:50 1998
Timezone is UTC+10.0
Password:
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 1.9.17p4]
security=user
smb: \> help
ls              dir             lcd             cd              pwd
get             mget            put             mput            rename
more            mask            del             rm              mkdir
md              rmdir           rd              pq              prompt
recurse         translate       lowercase       print           printmode
queue           qinfo           cancel          stat            quit
q               exit            newer           archive         tar
blocksize       tarmode         setmode         help            ?
```

```
!
smb: \> ls *.pdf
  ei010106.pdf                          129777  Mon Jan 26 12:34:06 1998
  ei020102.pdf                          229292  Mon Jan 26 12:34:54 1998
  ei020103.pdf                          291979  Mon Jan 26 12:35:22 1998

                50176 blocks of size 16384. 2963 blocks available
smb: \>
```

Once you connect with `smbclient` you see the `smbclient` prompt at which you can enter a number of commands. This acts a bit like a command-line `ftp` prompt.

Rather than use the fairly cumbersome smbclient interface to SMB drives Linux's virtual file system comes to the rescue. With the comment smbmount you can connect SMB drives from Windows machines to your Linux machine and use them as you would any other drive.

Exercise

16.11. Check that Samba is installed and configured on your system. Use `smbclient` or a Windows machine to see if you can connect to your home directory.

16.12. This chapter suggests that any network application can be broken down into ports, daemons, clients and protocols. Referring to the above discussion and other available documentation what are the ports, daemons, clients and protocols involved in using Samba on a Linux machine.

Email

Electronic mail, at least on the surface, looks fairly easy. However there are a number of issues that make configuring and maintaining Internet electronic mail a complex and occasionally frustrating task. Examining this task in-depth is beyond the scope of this subject. Instead, the following pages will provide an overview of the electronic mail system.

Email components

Programs that help send, reply and distribute email are divided into three categories

- mail user agents (MUA),
  These are the programs that people use to read and send email. Common MUAs include Eudora, Netscape (it has a mail and news reader as well as a Web browser) and text-based tools such as `elm` or `pine`. MUAs allow a user to read and write email.
- mail delivery agents (MDA),
  Once a mail message is delivered to the right computer, the MDA is responsible for placing it into the appropriate mail file.
- mail transport agents (MTA).
  Perform a number of tasks including some delivery, forwarding of email to

other MTAs closer to the final recipient and some address translation.

Figure 16.4 provides an overview of how these components fit together.

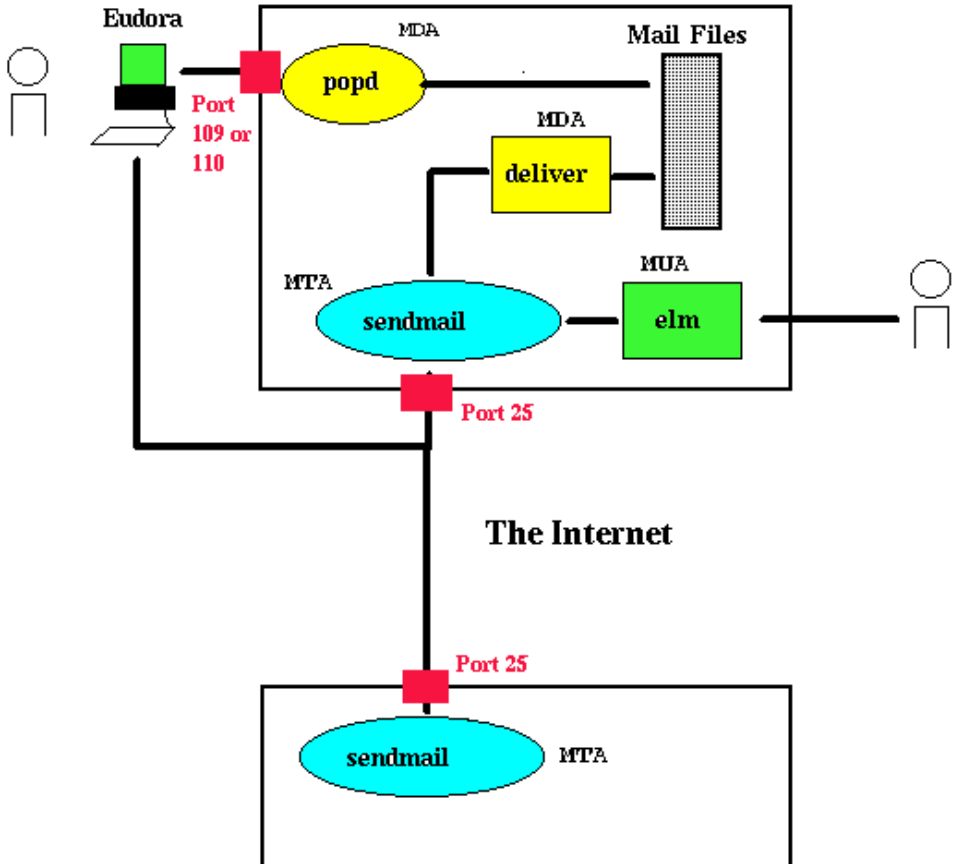


Figure 16.4
An overview of the mail system

The following is a brief description of how email is delivered for most people

- Mail daemon
  Most people will have an account on a mail server which will be running UNIX, Windows NT or some other operating system. At a minimum, the user's account will include a mail file. All email delivered for that user is appended onto the end of that mail file.
- Remote mail client
  Reading and writing mail for most people is done using a MUA like Eudora or Netscape on a remote mail client. This "remote mail client" is the user's normal computer they use for normal applications. The client mail computer will retrieve the user's mail from the mail server using a protocol such as POP or IMAP (see Table 16.6). Sending email will be via the SMTP protocol to the mail server's SMTP daemon (`sendmail` if it's the server is a UNIX computer).

Email Protocols

Table 16.7 lists some of the common protocols associated with email and briefly describes their purpose.

Protocol	Description
SMTP	Simple Mail Transport Protocol, the protocol used to transport mail from one Internet host to another
POP	Post Office Protocol, defines a method by which a small host can obtain mail from a larger host without running a MTA (like `sendmail`). Described in RFCs 1725 1734
IMAP	Internet Message Access Protocol, allows client mail programs to access and manipulate electronic mail messages on a server, including the manipulation of folders. Described in RFCs 1730, 1731.
MIME	Multipurpose Internet Mail Extensions, defines methods for sending binary data such as Word documents, pictures and sounds via Internet email which is distributed as text. Described in RFCs 1521 1522 and others.
PEM	Privacy-Enhanced Mail, message encryption and authentication procedures, proposed standard outlined in RFCs 1421, 1422 and 1423
Format of text messages	The standard format of Internet email which is described in RFC822

Table 16.7
Protocols and standards associated with Email

Unix mail software

Your RedHat Linux machine will include the following software related to email

- `sendmail`
  `sendmail` is the UNIX MTA. It may well be one of the most difficult and hated pieces of software in the world. However, recent versions have solved many of its problems. `sendmail` is the SMTP daemon on most UNIX machines. That is it is the server that handles SMTP requests.
- `popd`
  The pop daemon is contacted by MTAs such as Eudora when they wish to transfer a user's email from the server onto the client.
- `imapd`
  The imap daemon may not be installed on all machines but it is distributed with RedHat. `imapd` responds to MTAs which use imap to transfer email from the server to the client. The readings below contain a pointer to a document which describes the differences between IMAP and POP.
- various mail clients
  A RedHat machine will include a number of mail clients including `mutt`, `elm`, `pine`, `mh`, and Netscape.

Reading

The resource materials section on the 85321 Website/CD-ROM has pointers to a number of documents including a `sendmail` tutorial and a comparison of IMAP and POP. You will need to use these resources for the following exercise.

Exercises

16.13. Set up email on your Linux machine (refer to the Linux mail HOW-TO). Included in the procedure, obtain a POP mail client and get it working. The Netscape web browser includes a POP mail client for UNIX (it's what I use to read my mail).

16.14. The latest versions of Netscape also support IMAP. Configure your system to use IMAP rather than POP.

World-Wide Web

The World-Wide Web is the killer application which has really taken the Internet by storm. Most of the Web servers currently on the Internet are UNIX machines running the Apache Web server (http://www.apache.org/). RedHat comes with Apache pre-installed. If you use a Web browser to connect to your Linux machine (e.g. http://localhost/) Redhat provides pointers to documentation on configuring Apache.

Conclusions

This chapter has looked in general at how network services work and in particular at file and print sharing with Samba, email and World-Wide Web. Most network services consist of a

- port
- daemon
- client
- protocol.

Client programs communicate using an agreed upon protocol with daemons which await requests on a particular port.

Network ports are used to deliver information to one of the many network applications that may be running on a computer. Network ports from 0-1024 are used for pre-defined purposes. The allocation of those ports to applications is done in the `/etc/services` file. The `netstat` command can be used to examine the currently active network connections including which ports are being used.

Network daemons are either started in the system start-up scripts (`/etc/rc.d/*`) or by the `inetd` daemon. The file `/etc/inetd.conf` is used to configure which servers `inetd` will start.

Most Linux systems come already installed with `tcpd` (TCPWrappers). `tcpd` works with `inetd` to provide a number of additional features including logging, user validation and access control.

Intranets are the latest industry buzzword and are simply a local area network built using Internet protocols. Linux in conjunction with Samba and other public domain tools can act as a very cheap Intranet server offering file and print services, WWW server, electronic mail, `ftp` and other Internet services. Samba is a public domain piece of software that enables a UNIX computer to act as a file and printer server for client machines running Windows and other LanManager clients.

Review Questions

16.1

Explain the role each of the following play in UNIX networking

1) `/etc/services`
2) `/etc/inetd.conf`
3) `inetd`
4) `tcpd`

16.2

You've just obtained the daemon for WWWWW (the fictious replacement for the WWW). The daemon uses the protocol HTTTTTTP, wants to use port 81 and is likely to get many requests. Outline the steps you would have to complete to install the daemon including

- the files you would have to modify and why
- how you would start the daemon (it's a program called `htttttpd`)

16.3

People have been trying to `telnet` to your machine `server.my.domain`. List all the things that could be stopping them from logging in.

Chapter 17

Security

Local Introduction

The following reading is taken from the Security HOW-TO by Kevin Fenzi and Dave Wreski as part of the Linux Documentation Project. It offers a much better coverage of the material than the original, locally produced chapter (that material is available from the 85321 website/CD-ROM if you feel the need to do a comparison or want an alternative discussion of the data).

As you read through the following think about

- How the advice included here would change the way your personal Linux computer is currently configured?
- How this advice would change the way you managed a server in a small organisation?

The HOWTO itself mentions a wide range of other resources you can use to get more information about the topic of Security.

Linux Security HOWTO

Kevin Fenzi, kevin@scrye.com & Dave Wreski, dave@nic.com

v1.0.2, 25 April 1999

This document is a general overview of security issues that face the administrator of Linux systems. It covers general security philosophy and a number of specific examples of how to better secure your Linux system from intruders. Also included are pointers to security-related material and programs. Improvements, constructive criticism, additions and corrections are gratefully accepted. Please mail your feedback to both authors, with "Security HOWTO" in the subject.

Introduction

This document covers some of the main issues that affect Linux security. General philosophy and net-born resources are discussed.

A number of other HOWTO documents overlap with security issues, and those documents have been pointed to wherever appropriate.

This document is not meant to be a up to date exploits document. Large numbers of new exploits happen all the time. This document will tell you where to look for such up to date information, and will give some general methods to prevent such exploits from taking place.

New Versions of this Document

New versions of this document will be periodically posted to comp.os.linux.answers. They will also be added to the various anonymous FTP sites that archive such information, including:

ftp://metalab.unc.edu/pub/Linux/docs/HOWTO

In addition, you should generally be able to find this document on the Linux World Wide Web home page via:

http://metalab.unc.edu/mdw/linux.html

Finally, the very latest version of this document should also be available in various formats from:

http://scrye.com/~kevin/lsh/

Feedback

All comments, error reports, additional information and criticism of all sorts should be directed to:

kevin@scrye.com

and

dave@nic.com

Note: Please send your feedback to both authors. Also, be sure and include "Linux" "security", or "HOWTO" in your subject to avoid Kevin's spam filter.

Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and other content at your own risk. Additionally, this is an early version, possibly with many inaccuracies or errors.

A number of the examples and descriptions use the RedHat(tm) package layout and system setup. Your mileage may vary.

As far as we know, only programs that, under certain terms may be used or evaluated for personal purposes will be described. Most of the programs will be available, complete with source, under GNU <http://www.gnu.org/copyleft/gpl.html> terms.

Copyright Information

impose additional restrictions on its distribution. Exceptions to these rules may be granted under certain conditions; please contact the Linux HOWTO coordinator at the address given below.

- If you have questions, please contact Tim Bynum, the Linux HOWTO coordinator, at tjbynum@metalab.unc.edu

Overview

This document will attempt to explain some procedures and commonly-used software to help your Linux system be more secure. It is important to discuss some of the basic concepts first, and create a security foundation, before we get started.

Why Do We Need Security?

In the ever-changing world of global data communications, inexpensive Internet connections, and fast-paced software development, security is becoming more and more of an issue. Security is now a basic requirement because global computing is inherently insecure. As your data goes from point A to point B on the Internet, for example, it may pass through several other points along the way, giving other users the opportunity to intercept, and even alter, it. Even other users on your system may maliciously transform your data into something you did not intend. Unauthorized access to your system may be obtained by intruders, also known as "crackers", who then use advanced knowledge to impersonate you, steal information from you, or even deny you access to your own resources. If you're wondering what the difference is between a "Hacker" and a "Cracker", see Eric Raymond's document, "How to Become A Hacker", available at http://sagan.earthspace.net/~esr/faqs/hacker-howto.html.

How Secure Is Secure?

First, keep in mind that no computer system can ever be "completely secure". All you can do is make it increasingly difficult for someone to compromise your system. For the average home Linux user, not much is required to keep the casual cracker at bay. For high profile Linux users (banks, telecommunications companies, etc), much more work is required.

Another factor to take into account is that the more secure your system is, the more intrusive your security becomes. You need to decide where in this balancing act your system will still usable, and yet secure for your purposes. For instance, you could require everyone dialing into your system to use a call-back modem to call them back at their home number. This is more secure, but if someone is not at home, it makes it difficult for them to login. You could also setup your Linux system with no network or connection to the Internet, but this limits it's usefulness.

If you are a large to medium-sized site, you should establish a security policy stating how much security is required by your site and what auditing is in place to check it. You can find a well-known security policy example at http://ds.internic.net/rfc/rfc2196.txt. It has been recently updated, and contains a great framework for establishing a security policy for your company.

What Are You Trying to Protect?

Before you attempt to secure your system, you should determine what level of threat you have to protect against, what risks you should or should not take, and how vulnerable your system is as a result. You should analyze your system to know what you're protecting, why you're protecting it, what value it has, and who has responsibility for your data and other assets.

- Risk is the possibility that an intruder may be successful in attempting to access your computer. Can an intruder read or write files, or execute programs that could cause damage? Can they delete critical data? Can they prevent you or your company from getting important work done? Don't forget: someone gaining access to your account, or your system, can also impersonate you.

  Additionally, having one insecure account on your system can result in your entire network being compromised. If you allow a single user to login using a .rhosts file, or to use an insecure service, such as tftp, you risk an intruder getting 'his foot in the door'. Once the intruder has a user account on your system, or someone else's system, it can be used to gain access to another system, or another account.

- Threat is typically from someone with motivation to gain unauthorized access to your network or computer. You must decide who you trust to have access to your system, and what threat they could pose.

  There are several types of intruders, and it is useful to keep their different characteristics in mind as you are securing your systems.

  - The Curious - This type of intruder is basically interested in finding out what type of system and data you have.
  - The Malicious - This type of intruder is out to either bring down your systems, or deface your web page, or otherwise force you to spend time and money recovering from the damage he has caused.
  - The High-Profile Intruder - This type of intruder is trying to use your system to gain popularity and infamy. He might use your high-profile system to advertise his abilities.
  - The Competition - This type of intruder is interested in what data you have on your system. It might be someone who thinks you have something that could benefit him, financially or otherwise.
  - The Borrowers - This type of intruder is interested in setting up shop on your system and using it's resources for their own purposes. They typically will run chat or irc servers, porn archive sites, or even DNS servers.
  - The Leapfrogger - This type of intruder is only interested in your system to use it to get into other systems. If your system is well connected or a gateway to a number of internal hosts, you may well see this type trying to compromise your system.

Vulnerability describes how well-protected your computer is from another network, and the potential for someone to gain unauthorized access. What's at stake if someone breaks into your system? Of course the concerns of a dynamic PPP home user will be different from those of a company connecting

their machine to the Internet, or another large network.
How much time would it take to retrieve/recreate any data that was lost? An initial time investment now can save ten times more time later if you have to recreate data that was lost. Have you checked your backup strategy, and verified your data lately?

Developing A Security Policy

Create a simple, generic policy for your system that your users can readily understand and follow. It should protect the data you're safeguarding as well as the privacy of the users. Some things to consider adding are: who has access to the system (Can my friend use my account?), who's allowed to install software on the system, who owns what data, disaster recovery, and appropriate use of the system.

A generally accepted security policy starts with the phrase

That which is not permitted is prohibited

This means that unless you grant access to a service for a user, that user shouldn't be using that service until you do grant access. Make sure the policies work on your regular user account. Saying, "Ah, I can't figure this permissions problem out, I'll just do it as root" can lead to security holes that are very obvious, and even ones that haven't been exploited yet.

rfc1244 is a document that describes how to create your own network security polity.

rfc1281 is a document that shows an example security policy with detailed descriptions of each step.

Finally, you might want to look at the COAST policy archive at ftp://coast.cs.purdue.edu/pub/doc/policy to see what some real life security policies look like.

Means of Securing Your Site

This document will discuss various means with which you can secure the assets you have worked hard for: your local machine, your data, your users, your network, even your reputation. What would happen to your reputation if an intruder deleted some of your users' data? Or defaced your web site? Or published your company's corporate project plan for next quarter? If you are planning a network installation, there are many factors you must take into account before adding a single machine to your network.

Even if you have a single dialup PPP account, or just a small site, this does not mean intruders won't be interested in your systems. Large, high profile sites are not the only targets -- many intruders simply want to exploit as many sites as possible, regardless of their size. Additionally, they may use a security hole in your site to gain access to other sites you're connected to.

Intruders have a lot of time on their hands, and can avoid guessing how you've obscured your system just by trying all the possibilities. There are also a number of reasons an intruder may be interested in your systems, which we will discuss later.

Host Security

Perhaps the area of security on which administrators concentrate most is host-based security. This typically involves making sure your own system is secure, and hoping everyone else on your network does the same. Choosing good passwords, securing your host's local network services, keeping good accounting records, and upgrading programs with known security exploits are among the things the local security administrator is responsible for doing. Although this is absolutely necessary, it can become a daunting task once your network becomes larger than a few machines.

Network Security

Network security is also as necessary as local host security. With hundreds, thousands, or more computers on the same network, you can't rely on each one of those systems being secure. Ensuring that only authorized users can use your network, building firewalls, using strong encryption, and ensuring there are no "rogue" (that is, unsecured) machines on your network are all part of the network security administrator's duties.

This document will discuss some of the techniques used to secure your site, and hopefully show you some of the ways to prevent an intruder from gaining access to what you are trying to protect.

Security Through Obscurity

One type of security that must be discussed is "security through obscurity". This means, for example, moving a service that has known security vunerabilities to a non standard port in hopes that attackers won't notice it's there and thus won't exploit it. Rest assured that they can determine that its there and will exploit it. Security through obscurity is no security at all. Simply because you may have a small site, or a relatively low profile, does not mean an intruder won't be interested in what you have. We'll discuss what you're protecting in the next sections.

Organization of This Document

This document has been divided into a number of sections. They cover several broad security issues. The first, *Physical Security*, covers how you need to protect your physical machine from tampering. The second, *Local Security*, describes how to protect your system from tampering by local users. The third, *Files and Filesystem Security*, shows you how to setup your filesystems and premissions on your files. The next, *Password Security and Encryption*, discusses how to use encryption to better secure your machine and network. *Kernel Security* discusses what kernel options you should set or be aware of for a more secure system. *Network Security*, describes how to better secure your Linux system from network attacks. *Security Preparation*, discusses how to prepare your machine(s) before bringing them on-line. Next, *What To Do During and After a Break-in*, discusses what to do when you detect a system compromise in progress or detect one that has recently happened. In *Security Resources*, some primary security resources are enumerated. The Q and A section *Frequently Asked Questions*, answers some frequently asked questions, and finally a conclusion in *Conclusion* section.

The two main points to realize when reading this document are:

- Be aware of your system. Check system logs such as `/var/log/messages` and keep an eye on your system, and
- Keep your system up to date by making sure you have installed the current versions of software and have upgraded per security alerts. Just doing this will help make your system markedly more secure.

Physical Security

The first layer of security you need to take into account is the physical security of your computer systems. Who has direct physical access to your machine? Should they? Can you protect your machine from their tampering? Should you?

How much physical security you need on your system is very dependent on your situation, and/or budget.

If you are a home user, you probably don't need a lot (although you might need to protect your machine from tampering by children or annoying relatives). If you are in a Lab, you need considerably more, but users will still need to be able to get work done on the machines. Many of the following sections will help out. If you are in an office, you may or may not need to secure your machine off hours or while you are away. At some companies, leaving your console unsecured is a termination offense.

Obvious physical security methods such as locks on doors, cables, locked cabinets, and video surveillance are all good ideas, but beyond the scope of this document. :)

Computer locks

Many modern PC cases include a "locking" feature. Usually this will be a socket on the front of the case that allows you to turn an included key to a locked or unlocked position. Case locks can help prevent someone from stealing your PC, or opening up the case and directly manipulating/stealing your hardware. They can also sometimes prevent someone from rebooting your computer on their own floppy or other hardware.

These case locks do different things according to the support in the motherboard and how the case is constructed. On many PC's they make it so you have to break the case to get the case open. On some others, they make it so that it will not let you plug in new keyboards and mice. Check your motherboard or case instructions for more information. This can sometimes be a very useful feature, even though the locks are usually very low quality and can easily be defeated by attackers with locksmithing.

Some cases (most notably SPARCs and macs) have a dongle on the back that, if you put a cable through attackers would have to cut the cable or break the case to get into it. Just putting a padlock or combo lock through these can be a good deterrent to someone stealing your machine.

BIOS Security

The BIOS is the lowest level of software that configures or manipulates your x86-based hardware. LILO and other Linux boot methods access the BIOS to determine how to boot up your Linux machine. Other hardware that Linux runs on has similar software (OpenFirmware on Macs and new Suns, Sun boot PROM, etc...). You can use your BIOS to prevent attackers from rebooting your machine and manipulating your Linux system.

Many PC BIOSs let you set a boot password. This doesn't provide all that much security (the BIOS can be reset, or removed if someone can get into the case), but might be a good deterrent (i.e. it will take time and leave traces of tampering). Similarly, on S/Linux (Linux for SPARC(tm) porcessor machines), your EEPROM can be set to require a boot-up password. This might slow attackers down.

Many x86 BIOSs also allow you to specify various other good security settings. Check your BIOS manual or look at it the next time you boot up. For example, some BIOSs disallow booting from floppy drives and some require passwords to access some BIOS features.

Note: If you have a server machine, and you set up a boot password, your machine will not boot up unattended. Keep in mind that you will need to come in and supply the password in the event of a power failure. ;(

Boot Loader Security

The various Linux boot loaders also can have a boot password set. LILO, for example, has password and restricted settings; password always requires password at boot time, whereas restricted requires a boot-time password only if you specify options (such as single) at the LILO prompt.

Keep in mind when setting all these passwords that you need to remember them. :) Also remember that these passwords will merely slow the determined attacker. They won't prevent someone from booting from a floppy, and mounting your root partition. If you are using security in conjunction with a boot loader, you might as well disable booting from a floppy in your computer's BIOS, and password-protect the BIOS.

If anyone has security-related information from a different boot loader, we would love to hear it. (grub, silo, milo, linload, etc).

Note: If you have a server machine, and you set up a boot password, your machine will not boot up unattended. Keep in mind that you will need to come in and supply the password in the event of a power failure. ;(

xlock and vlock

If you wander away from your machine from time to time, it is nice to be able to "lock" your console so that no one tampers with or looks at your work. Two programs that do this are: xlock and vlock.

xlock is a X display locker. It should be included in any Linux distributions that support X. Check out the man page for it for more options, but in general you can run xlock from any xterm on your console and it will lock the display and require your password to unlock.

vlock is a simple little program that allows you to lock some or all of the virtual consoles on your Linux box. You can lock just the one you are working in or all of them. If you just lock one, others can come in and use the console; they will just not be able to use your virtual console until you unlock it. vlock ships with redhat Linux, but your mileage may vary.

Of course locking your console will prevent someone from tampering with your work, but won't prevent them from rebooting your machine or otherwise disrupting your work. It also does not prevent them from accessing your machine from another machine on the network and causing problems.

More importantly, it does not prevent someone from switching out of the X Window System entirely, and going to a normal virtual console login prompt, or to the VC that X11 was started from, and suspending it, thus obtaining your priviledges. For this reason, you might consider only using it while under control of xdm.

Detecting Physical Security Compromises

The first thing to always note is when your machine was rebooted. Since Linux is a robust and stable OS, the only times your machine should reboot is when you take it down for OS upgrades, hardware swapping, or the like. If your machine has rebooted without you doing it, that may be a sign that an intruder has compromised it. Many of the ways that your machine can be compromised require the intruder to reboot or power off your machine.

Check for signs of tampering on the case and computer area. Although many intruders clean traces of their presence out of logs, it's a good idea to check through them all and note any discrepancy.

It is also a good idea to store log data at a secure location, such as a dedicated log server within your well-protected network. Once a machine has been compromised, log data becomes of little use as it most likely has also been modified by the intruder.

The syslog daemon can be configured to automatically send log data to a central syslog server, but this is typically sent in cleartext data, allowing an intruder to view data as it is being transferred. This may reveal information about your network that is not intended to be public. There are syslog daemons available that encrypt the data as it is being sent.

Also be aware that faking syslog messages is easy - with an exploit program having been published. Syslog even accepts net log entries claiming to come from the local host without indicating their true origin.

Some things to check for in your logs:

- Short or incomplete logs.
- Logs containing strange timestamps.
- Logs with incorrect permissions or ownership.
- Records of reboots or restarting of services.
- missing logs.
- su entries or logins from strange places.

We will discuss system log data ‘‘later” in the HOWTO.

Local Security

The next thing to take a look at is the security in your system against attacks from local users. Did we just say local users? Yes!

Getting access to a local user account is one of the first things that system intruders attempt while on their way to exploiting the root account. With lax local security, they can then "upgrade" their normal user access to root access using a variety of bugs and poorly setup local services. If you make sure your local security is tight, then the intruder will have another hurdle to jump.

Local users can also cause a lot of havoc with your system even (especially) if they really are who they say they are. Providing accounts to people you don't know or have no contact information for is a very bad idea.

Creating New Accounts

You should make sure to provide user accounts with only the minimal requirements for the task they need to do. If you provide your son (age 10) with an account, you might want him to only have access to a word processor or drawing program, but be unable to delete data that is not his.

Several good rules of thumb when allowing other people legitimate access to your Linux machine:

- Give them the minimal amount of privileges they need.
- Be aware when/where they login from, or should be logging in from.
- Make sure to remove inactive accounts
- The use of the same user-ID on all computers and networks is advisable to ease account maintence, as well as permit easier analysis of log data.
- The creation of group user-IDs should be absolutely prohibited.
- User accounts also provide accountability, and this is not possible with group accounts.

Many local user accounts that are used in security compromises are ones that have not been used in months or years. Since no one is using them they, provide the ideal attack vehicle.

Root Security

The most sought-after account on your machine is the root (superuser) account. This account has authority over the entire machine, which may also include authority over other machines on the network. Remember that you should only use the root account for very short, specific tasks, and should mostly run as a normal user. Even small mistakes made while logged in as the root user can cause problems. The less time you are on with root privledges, the safer you will be.

Several tricks to avoid messing up your own box as root:

- When doing some complex command, try running it first in a non-destructive way...especially commands that use globbing: e.g., if you want to do "rm foo*.bak", first do "ls foo*.bak" and make sure you are going to

delete the files you think you are. Using echo in place of destructive commands also sometimes works.

- Provide your users with a default alias to the rm command to ask for confirmation for deletion of files.
- Only become root to do single specific tasks. If you find yourself trying to figure out how to do something, go back to a normal user shell until you are sure what needs to be done by root.
- The command path for the root user is very important. The command path (that is, the PATH environment variable) specifies the directories in which the shell searches for programs. Try to limit the command path for the root user as much as possible, and never include . (which means "the current directory") in your PATH. Additionally, never have writable directories in your search path, as this can allow attackers to modify or place new binaries in your search path, allowing them to run as root the next time you run that command.
- Never use the rlogin/rsh/rexec suite of tools (called the r- utilities) as root. They are subject to many sorts of attacks, and are downright dangerous run as root. Never create a .rhosts file for root.
- The /etc/securetty file contains a list of terminals that root can login from. By default (on Red Hat Linux) this is set to only the local virtual consoles(vtys). Be very careful of adding anything else to this file. You should be able to login remotely as your regular user account and then su if you need to (hopefully over `ssh` or other encrypted channel), so there is no need to be able to login directly as root.
- Always be slow and deliberate running as root. Your actions could affect a lot of things. Think before you type!

If you absolutely positively need to allow someone (hopefully very trusted) to have root access to your machine, there are a few tools that can help. sudo allows users to use their password to access a limited set of commands as root. This would allow you to, for instance, let a user be able to eject and mount removable media on your Linux box, but have no other root privileges. sudo also keeps a log of all successful and unsuccessful sudo attempts, allowing you to track down who used what command to do what. For this reason sudo works well even in places where a number of people have root access, because it helps you keep track of changes made.

Although sudo can be used to give specific users specific privileges for specific tasks, it does have several shortcomings. It should be used only for a limited set of tasks, like restarting a server, or adding new users. Any program that offers a shell escape will give root access to a user invoking it via sudo. This includes most editors, for example. Also, a program as innocuous as /bin/cat can be used to overwrite files, which could allow root to be exploited. Consider sudo as a means for accountability, and don't expect it to replace the root user and still be secure.

Files and Filesystem Security

A few minutes of preparation and planning ahead before putting your systems online can help to protect them and the data stored on them.

- There should never be a reason for users' home directories to allow SUID/SGID programs to be run from there. Use the nosuid option in /etc/fstab for partitions that are writable by others than root. You may also wish to use nodev and noexec on users' home partitions, as well as /var, thus prohibiting execution of programs, and creation of character or block devices, which should never be necessary anyway.
- If you are exporting filesystems using NFS, be sure to configure /etc/exports with the most restrictive access possible. This means not using wildcards, not allowing root write access, and exporting read-only wherever possible.
- Configure your users' file-creation umask to be as restrictive as possible. See ‘‘umask settings”.
- If you are mounting filesystems using a network filesystem such as NFS, be sure to configure /etc/exports with suitable restrictions. Typically, using ‘nodev’, ‘nosuid’, and perhaps ‘noexec’, are desireable.
- Set filesystem limits instead of allowing unlimited as is the default. You can control the per-user limits using the resource-limits PAM module and /etc/pam.d/limits.conf. For example, limits for group users might look like this:

```
@users     hard  core    0
@users     hard  nproc   50
@users     hard  rss     5000
```

  This says to prohibit the creation of core files, restrict the number of processes to 50, and restrict memory usage per user to 5M.
- The /var/log/wtmp and /var/run/utmp files contain the login records for all users on your system. Their integrity must be maintained because it can be used to determine when and from where a user (or potential intruder) has entered your system. These files should also have 644 permissions, without affecting normal system operation.
- The immutable bit can be used to prevent accidentally deleting or overwriting a file that must be protected. It also prevents someone from creating a symbolic link to the file (such symbolic links have been the source of attacks involving deleting /etc/passwd or /etc/shadow). See the chattr(1) man page for information on the immutable bit.
- SUID and SGID files on your system are a potential security risk, and should be monitored closely. Because these programs grant special privileges to the user who is executing them, it is necessary to ensure that insecure programs are not installed. A favorite trick of crackers is to exploit SUID-root programs, then leave a SUID program as a backdoor to get in the next time, even if the original hole is plugged. Find all SUID/SGID programs on your system, and keep track of what they are, so you are aware of any changes which could indicate a potential intruder. Use the following command to find all SUID/SGID programs on your system:

```
root#  find / -type f \( -perm -04000 -o -perm -02000 \)
```

The Debian distribution runs a job each night to determine what SUID files exist. It then compairs this to the previous nights run. You can look in /var/log/suid* for this log.

You can remove the SUID or SGID permissions on a suspicious program with chmod, then change it back if you absolutely feel it is necessary.

- World-writable files, particularly system files, can be a security hole if a cracker gains access to your system and modifies them. Additionally, world-writable directories are dangerous, since they allow a cracker to add or delete files as he wishes. To locate all world-writable files on your system, use the following command:

  ```
  root# find / -perm -2 ! -type l –ls
  ```

  and be sure you know why those files are writable. In the normal course of operation, several files will be world-writable, including some from /dev, and symbolic links, thus the ! -type l which excludes these from the previous find command.

- Unowned files may also be an indication an intruder has accessed your system. You can locate files on your system that have no owner, or belong to no group with the command:

  ```
  root# find / -nouser -o -nogroup -print
  ```

- Finding .rhosts files should be a part of your regular system administration duties, as these files should not be permitted on your system. Remember, a cracker only needs one insecure account to potentially gain access to your entire network. You can locate all .rhosts files on your system with the following command:

  ```
  root# find /home -name .rhosts -print
  ```

- Finally, before changing permissions on any system files, make sure you understand what you are doing. Never change permissions on a file because it seems like the easy way to get things working. Always determine why the file has that permission before changing it.

Umask Settings

The umask command can be used to determine the default file creation mode on your system. It is the octal complement of the desired file mode. If files are created without any regard to their permissions settings, the user could inadvertently give read or write permission to someone that should not have this permission. Typically umask settings include 022, 027, and 077 (which is the most restrictive). Normally the umask is set in /etc/profile, so it applies to all users on the system. The file creation mask can be calculated by subtracting the desired value from 777. In other words, a umask of 777 would cause newly-created files to contain no read, write or execute permission for anyone. A mask of 666 would cause newly-created files to have a mask of 111. For example, you may have a line that looks like this:

```
# Set the user’s default umask
umask 033
```

Be sure to make root's umask 077, which will disable read, write, and execute permission for other users, unless explicitly changed using chmod. In this case, newly-created directories would have 744 permissions, obtained by subtracting 033 from 777. Newly-created files using the 033 umask would have permissions of 644.

If you are using Red Hat, and adhere to their user and group ID creation scheme (User Private Groups), it is only necessary to use 002 for a umask. This is due to the fact that the default configuration is one user per group.

File Permissions

It's important to ensure that your system files are not open for casual editing by users and groups who shouldn't be doing such system maintenance.

Unix seperates access control on files and directories according to three characteristics: owner, group, and other. There is always exactly one owner, any number of members of the group, and everyone else.

A quick explanation of Unix permissions:

Ownership - Which user(s) and group(s) retain(s) control of the permission settings of the node and parent of the node

Permissions - Bits capable of being set or reset to allow certain types of access to it. Permissions for directories may have a different meaning than the same set of permissions on files.

Read:

- To be able to view contents of a file
- To be able to read a directory

Write:

- To be able to add to or change a file
- To be able to delete or move files in a directory

Execute:

To be able to run a binary program or shell script

To be able to search in a directory, combined with read permission

Save Text Attribute: (For directories)

The "sticky bit" also has a different meaning when applied to directories than when applied to files. If the sticky bit is set on a directory, then a user may only delete files that the he owns or for which he has explicit write permission granted, even when he has write access to the directory. This is designed for directories like /tmp, which are world-writable, but where it may not be desirable to allow any user to delete files at will. The sticky bit is seen as a t in a long directory listing.

SUID Attribute: (For Files)

This describes set-user-id permissions on the file. When the set user ID access mode is set in the owner permissions, and the file is executable, processes which run it are granted access to system resources based on user who owns the file, as opposed to the user who created the process. This is the cause of many "buffer overflow" exploits.

SGID Attribute: (For Files)

If set in the group permissions, this bit controls the "set group id" status of a file. This behaves the same way as SUID, except the group is affected instead. The file must be executable for this to have any effect.

SGID Attribute: (For directories)

If you set the SGID bit on a directory (with chmod g+s directory), files created in that directory will have their group set to the directory's group.

You - The owner of the file

Group - The group you belong to

Everyone - Anyone on the system that is not the owner or a member of the group

File Example:

```
-rw-r--r--  1 kevin  users         114 Aug 28  1997 .zlogin
1st bit - directory?             (no)
 2nd bit - read by owner?         (yes, by kevin)
  3rd bit - write by owner?        (yes, by kevin)
   4th bit - execute by owner?      (no)
    5th bit - read by group?         (yes, by users)
     6th bit - write by group?        (no)
      7th bit - execute by group?      (no)
       8th bit - read by everyone?      (yes, by everyone)
        9th bit - write by everyone?     (no)
         10th bit - execute by everyone?  (no)
```

The following lines are examples of the minimum sets of permissions that are required to perform the access described. You may want to give more permission than what's listed here, but this should describe what these minimum permissions on files do:

```
-r--------  Allow read access to the file by owner
--w-------  Allows the owner to modify or delete the file
            (Note that anyone with write permission to the directory
            the file is in can overwrite it and thus delete it)
---x------  The owner can execute this program, but not shell
            scripts, which still need read permission
---s------  Will execute with effective User ID = to owner
--------s-  Will execute with effective Group ID = to group
-rw------T  No update of "last modified time".  Usually used for swap
            files
---t------  No effect.  (formerly sticky bit)
```

Directory Example:

```
drwxr-xr-x  3 kevin  users         512 Sep 19 13:47 .public_html/
1st bit - directory?             (yes, it contains many files)
 2nd bit - read by owner?         (yes, by kevin)
  3rd bit - write by owner?        (yes, by kevin)
   4th bit - execute by owner?      (yes, by kevin)
    5th bit - read by group?         (yes, by users
     6th bit - write by group?        (no)
      7th bit - execute by group?      (yes, by users)
       8th bit - read by everyone?      (yes, by everyone)
        9th bit - write by everyone?     (no)
         10th bit - execute by everyone?  (yes, by everyone)
```

The following lines are examples of the minimum sets of permissions that are required to perform the access described. You may want to give more permission than what's listed, but this should describe what these minimum permissions on directories do:

```
dr--------  The contents can be listed, but file attributes can't be
            read
d--x------  The directory can be entered, and used in full execution
            paths
dr-x------  File attributes can be read by owner
d-wx------  Files can be created/deleted, even if the directory
            isn't the current one
d------x-t  Prevents files from deletion by others with write
            access. Used on /tmp
d---s--s--  No effect
```

System configuration files (usually in /etc) are usually mode 640 (-rw-r-----), and owned by root. Depending on your sites security requirements, you might adjust this. Never leave any system files writable by a group or everyone. Some configuration files, including /etc/shadow, should only be readable by root, and directories in /etc should at least not be accessible by others.

SUID Shell Scripts

SUID shell scripts are a serious security risk, and for this reason the kernel will not honor them. Regardless of how secure you think the shell script is, it can be exploited to give the cracker a root shell.

Integrity Checking with Tripwire Tripwire

Another very good way to detect local (and also network) attacks on your system is to run an integrity checker like Tripwire. Tripwire runs a number of checksums on all your important binaries and config files and compares them against a database of former, known-good values as a reference. Thus, any changes in the files will be flagged.

It's a good idea to install Tripwire onto a floppy, and then physically set the write protect on the floppy. This way intruders can't tamper with Tripwire itself or change the database. Once you have Tripwire setup, it's a good idea to run it as part of your normal security administration duties to see if anything has changed.

You can even add a crontab entry to run Tripwire from your floppy every night and mail you the results in the morning. Something like:

```
# set mailto
MAILTO=kevin
```

```
# run Tripwire
15 05 * * * root /usr/local/adm/tcheck/tripwire
```

will mail you a report each morning at 5:15am.

Tripwire can be a godsend to detecting intruders before you would otherwise notice them. Since a lot of files change on the average system, you have to be careful what is cracker activity and what is your own doing.

You can find Tripwire at http://www.tripwiresecurity.com, free of charge. Manuals and support can be purchased.

5.4. Trojan Horses

"Trojan Horses" are named after the fabled ploy in Homer's "Iliad". The idea is that a cracker distributes a program or binary that sounds great, and encourages other people to download it and run it as root. Then the program can compromise their system while they are not paying attention. While they think the binary they just pulled down does one thing (and it might very well), it also compromises their security.

You should take care of what programs you install on your machine. Redhat provides MD5 checksums and PGP signatures on it's RPM files so you can verify you are installing the real thing. Other distributions have similar methods. You should never run any unfamiliar binary, for which you don't have the source, as root! Few attackers are willing to release source code to public scrutiny.

Although it can be complex, make sure you are getting the source for a program from its real distribution site. If the program is going to run as root, make sure either you or someone you trust has looked over the source and verified it.

Password Security and Encryption

One of the most important security features used today are passwords. It is important for both you and all your users to have secure, unguessable passwords. Most of the more recent Linux distributions include passwd programs that do not allow you to set a easily guessable password. Make sure your passwd program is up to date and has these features.

In-depth discussion of encryption is beyond the scope of this document, but an introduction is in order. Encryption is very useful, possibly even necessary in this day and age. There are all sorts of methods of encrypting data, each with its own set of characteristics.

Most Unicies (and Linux is no exception) primarily use a one-way encryption algorithm, called DES (Data Encryption Standard) to encrypt your passwords. This encrypted password is then stored in (typically) /etc/passwd (or less commonly) /etc/shadow. When you attempt to login, the password you type in is encrypted again and compared with the entry in the file that stores your passwords. If they match, it must be the same password, and you are allowed access. Although DES is a two-way encryption algorithm (you can code and then decode a message, given the right keys), the variant that most unices use is one-way. This means that it should not be possible to reverse the encryption to get the password from the contents of /etc/passwd (or /etc/shadow).

Brute force attacks, such as "Crack" or "John the Ripper" (see the later section covering these) can often guess passwords unless your password is sufficiently random. PAM modules (see below) allow you to use a different encryption routine with your passwords (MD5 or the like). You can use Crack to your advantage, as well. Consider periodically running Crack against your own password database, to find insecure passwords. Then contact the offending user, and instruct him to change his password.

You can go to http://consult.cern.ch/writeup/security/security_3.html for information on how to choose a good password.

PGP and Public-Key Cryptography

Public-key cryptography, such as that used for PGP, uses one key for encryption, and one key for decryption. Traditional cryptography, however, uses the same key for encryption and decryption; this key must be known to both parties, and thus somehow transferred from one to the other securely.

To alleviate the need to securely transmit the encryption key, public-key encryption uses two separate keys: a public key and a private key. Each person's public key is available by anyone to do the encryption, while at the same time each person keeps his or her private key to decrypt messages encrypted with the correct public key.

There are advantages to both public key and private key cryptography, and you can read about those differences in the RSA Cryptography FAQ <http://www.rsa.com/rsalabs/newfaq/>, listed at the end of this section.

PGP (Pretty Good Privacy) is well-supported on Linux. Versions 2.6.2 and 5.0 are known to work well. For a good primer on PGP and how to use it, take a look at the PGP FAQ: http://www.pgp.com/service/export/faq/55faq.cgi

Be sure to use the version that is applicable to your country. Due to export restrictions by the US Government, strong-encryption is prohibited from being transferred in electronic form outside the country.

US export controls are now managed by EAR (Export Administration Regulations). They are no longer governed by ITAR. There is also a step-by-step guide for configuring PGP on Linux available at

http://mercury.chem.pitt.edu/~angel/LinuxFocus/English/November1997/article7.html.

It was written for the international version of PGP, but is easily adaptable to the United States version. You may also need a patch for some of the latest versions of Linux; the patch is available at ftp://metalab.unc.edu/pub/Linux/apps/crypto.

There is a project working on a free re-implementation of pgp with open source. GnuPG is a complete and free replacement for PGP. Because it does not use IDEA or RSA it can be used without any restrictions. GnuPG is nearly in compliance with RFC2440 (OpenPGP). See the GNU Privacy Guard web page for more information: http://www.gpg.org/.

More information on cryptography can be found in the RSA cryptography FAQ, available at http://www.rsa.com/rsalabs/newfaq/. Here you will find information on such terms as "Diffie-Hellman", "public-key cryptography", "digital certificates", etc.

SSL, S-HTTP, HTTPS and S/MIME

Often users ask about the differences between the various security and encryption protocols, and how to use them. While this isn't an encryption document, it is a good idea to explain briefly what each protocol is, and where to find more information.

- SSL: - SSL, or Secure Sockets Layer, is an encryption method developed by Netscape to provide security over the Internet. It supports several different encryption protocols, and provides client and server authentication. SSL operates at the transport layer, creates a secure encrypted channel of data, and thus can seamlessly encrypt data of many types. This is most commonly seen when going to a secure site to view a secure online document with Communicator, and serves as the basis for secure communications with Communicator, as well as many other Netscape Communications data encryption. More information can be found at http://www.consensus.com/security/ssl-talk-faq.html. Information on Netscape's other security implementations, and a good starting point for these protocols is available at http://home.netscape.com/info/security-doc.html.
- S-HTTP: - S-HTTP is another protocol that provides security services across the Internet. It was designed to provide confidentiality, authentication, integrity, and non-repudiability [cannot be mistaken for someone else] while supporting multiple key-management mechanisms and cryptographic algorithms via option negotiation between the parties involved in each transaction. S-HTTP is limited to the specific software that is implementing it, and encrypts each message individually. [From RSA Cryptography FAQ, page 138]
- S/MIME: - S/MIME, or Secure Multipurpose Internet Mail Extension, is an encryption standard used to encrypt electronic mail and other types of messages on the Internet. It is an open standard developed by RSA, so it is likely we will see it on Linux one day soon. More information on S/MIME can be found at http://home.netscape.com/assist/security/smime/overview.html.

Linux IPSEC Implementations

Along with CIPE, and other forms of data encryption, there is also several other implementations of IPSEC for Linux. IPSEC is an effort by the IETF to create cryptographically-secure communications at the IP network level, and to provide authentication, integrity, access control, and confidentiality. Information on IPSEC and Internet draft can be found at http://www.ietf.org/html.charters/ipsec-charter.html. You can also find links to other protocols involving key management, and an IPSEC mailing list and archives.

The x-kernel Linux implementation, which is being developed at the University of Arizona, uses an object-based framework for implementing network protocols called x-kernel, and can be found at http://www.cs.arizona.edu/xkernel/hpcc-blue/linux.html. Most simply, the x-kernel is a method of passing messages at the kernel level, which makes for an easier implementation.

Another freely-available IPSEC implementation is the Linux FreeS/WAN IPSEC. Their web page states,

> "These services allow you to build secure tunnels through untrusted networks. Everything passing through the untrusted net is encrypted by the IPSEC gateway machine and decrypted by the gateway at the other end. The result is Virtual Private Network or VPN. This is a network which is effectively private even though it includes machines at several diffeent sites connected by the insecure Internet."

It's available for download from http://www.xs4all.nl/~freeswan/, and has just reached 1.0 at the time of this writing.

As with other forms of cryptography, it is not distributed with the kernel by default due to export restrictions.

ssh (Secure Shell) and stelnet

ssh and stelnet are programs that allow you to login to remote systems and have a encrypted connection.

ssh is a suite of programs used as a secure replacement for rlogin, rsh and rcp. It uses public-key cryptography to encrypt communications between two hosts, as well as to authenticate users. It can be used to securely login to a remote host or copy data between hosts, while preventing man-in-the-middle attacks (session hijacking) and DNS spoofing. It will perform data compression on your connections, and secure X11 communications between hosts. The ssh home page can be found at http://www.cs.hut.fi/ssh/

You can also use ssh from your Windows workstation to your Linux ssh server. There are several freely available Windows client implementations, including the one at http://guardian.htu.tuwien.ac.at/therapy/ssh/ as well as a commercial implementation from DataFellows, at http://www.datafellows.com. There is also a open source project to re-implement ssh called "psst...". For more information see: http://www.net.lut.ac.uk/psst/

SSLeay is a free implementation of Netscape's Secure Sockets Layer protocol, developed by Eric Young. It includes several applications, such as Secure telnet, a module for Apache, several databases, as well as several algorithms including DES, IDEA and Blowfish.

Using this library, a secure telnet replacement has been created that does encryption over a telnet connection. Unlike SSH, stelnet uses SSL, the Secure Sockets Layer protocol developed by Netscape. You can find Secure telnet and Secure FTP by starting with the SSLeay FAQ, available at http://www.psy.uq.oz.au/~ftp/Crypto/.

SRP is another secure telnet/ftp implementation. From their web page:

> "The SRP project is developing secure Internet software for free worldwide use. Starting with a fully-secure Telnet and FTP distribution, we hope to supplant weak networked authentication systems with strong replacements that do not sacrifice user-friendliness for security. Security should be the default, not an option!"

For more information, go to http://srp.stanford.edu/srp.

PAM - Pluggable Authentication Modules

Newer versions of the Red Hat Linux distribution ship with a unified authentication scheme called "PAM". PAM allows you to change your authentication methods and requirements on the fly, and encapsulate all local authentication methods without recompiling any of your binaries. Configuration of PAM is beyond the scope of this document, but be sure to take a look at the PAM web site for more information. http://www.kernel.org/pub/linux/libs/pam/index.html.

Just a few of the things you can do with PAM:

- Use encryption other than DES for your passwords. (Making them harder to brute-force decode)
- Set resource limits on all your users so they can't perform denial-of-service attacks (number of processes, amount of memory, etc)
- Enable shadow passwords (see below) on the fly
- allow specific users to login only at specific times from specific places

Within a few hours of installing and configuring your system, you can prevent many attacks before they even occur. For example, use PAM to disable the system-wide usage of .rhosts files in user's home directories by adding these lines to /etc/pam.d/rlogin:

```
#
# Disable rsh/rlogin/rexec for users
#
login auth required pam_rhosts_auth.so no_rhosts
```

Cryptographic IP Encapsulation (CIPE)

The primary goal of this software is to provide a facility for secure (against eavesdropping, including traffic analysis, and faked message injection) subnetwork interconnection across an insecure packet network such as the Internet.

CIPE encrypts the data at the network level. Packets traveling between hosts on the network are encrypted. The encryption engine is placed near the driver which sends and receives packets.

This is unlike SSH, which encrypts the data by connection, at the socket level. A logical connection between programs running on different hosts is encrypted.

CIPE can be used in tunnelling, in order to create a Virtual Private Network. Low-level encryption has the advantage that it can be made to work transparently between the two networks connected in the VPN, without any change to application software.

Summarized from the CIPE documentation:

> The IPSEC standards define a set of protocols which can be used (among other things) to build encrypted VPNs. However, IPSEC is a rather heavyweight and complicated protocol set with a lot of options, implementations of the full protocol set are still rarely used and some issues (such as key management) are still not fully resolved. CIPE uses a simpler approach, in which many things which can be parameterized (such as the

choice of the actual encryption algorithm used) are an install-time fixed choice. This limits flexibility, but allows for a simple (and therefore efficient, easy to debug...) implementation.

Further information can be found at
http://www.inka.de/~bigred/devel/cipe.html

As with other forms of cryptography, it is not distributed with the kernel by default due to export restrictions.

Kerberos

Kerberos is an authentication system developed by the Athena Project at MIT. When a user logs in, Kerberos authenticates that user (using a password), and provides the user with a way to prove her identity to other servers and hosts scattered around the network.

This authentication is then used by programs such as rlogin to allow the user to login to other hosts without a password (in place of the .rhosts file). This authentication method can also used by the mail system in order to guarantee that mail is delivered to the correct person, as well as to guarantee that the sender is who he claims to be.

Kerberos and the other programs that come with it, prevent users from "spoofing" the system into believing they are someone else. Unfortunately, installing Kerberos is very intrusive, requiring the modification or replacement of numerous standard programs.

You can find more information about kerberos by looking at the kerberos FAQ, and the code can be found at http://nii.isi.edu/info/kerberos/.

[From: Stein, Jennifer G., Clifford Neuman, and Jeffrey L. Schiller. "Kerberos: An Authentication Service for Open Network Systems." USENIX Conference Proceedings, Dallas, Texas, Winter 1998.]

Kerberos should not be your first step in improving security of your host. It is quite involved, and not as widely used as, say, SSH.

Shadow Passwords.

Shadow passwords are a means of keeping your encrypted password information secret from normal users. Normally, this encrypted passwords are stored in /etc/passwd file for all to read. Anyone can then run password guesser programs on them and attempt to determine what they are. Shadow passwords, by contrast, are saved in /etc/shadow, which only privileged users can read. In order to use shadow passwords, you need to make sure all your utilities that need access to password information are recompiled to support them. PAM (above) also allows you to just plug in a shadow module; it doesn't require re-compilation of executables. You can refer to the Shadow-Password HOWTO for further information if necessary. It is available at http://metalab.unc.edu/LDP/HOWTO/Shadow-Password-HOWTO.html It is rather dated now, and will not be required for distributions supporting PAM.

“Crack" and "John the Ripper"

If for some reason your passwd program is not enforcing hard-to-guess passwords, you might want to run a password-cracking program and make sure your users’ passwords are secure.

Password cracking programs work on a simple idea: they try every word in the dictionary, and then variations on those words, encrypting each one and checking it against your encrypted password. If they get a match they know what your password is.

There are a number of programs out there...the two most notable of which are "Crack" and "John the Ripper" (http://www.false.com/security/john/index.html) . They will take up a lot of your cpu time, but you should be able to tell if an attacker could get in using them by running them first yourself and notifying users with weak passwords. Note that an attacker would have to use some other hole first in order to read your /etc/passwd file, but such holes are more common than you might think.

Because security is only as strong as the most insecure host, it is worth mentioning that if you have any Windows machines on your network, you should check out L0phtCrack, a Crack implementation for Windows. It’s available from http://www.l0pht.com

CFS & TCFS: Cryptographic File Systems

CFS is a way of encrypting an entire directory trees and allowing users to store encrypted files on them. It uses a NFS server running on the local machine. RPMS are available at http://www.replay.com/redhat/, and more information on how it all works is at ftp://ftp.research.att.com/dist/mab/.

TCFS improves on CFS by adding more integration with the file system, so that it’s transparent to users that the file system that is encrypted. more information at: http://edu-gw.dia.unisa.it/tcfs/.

It also need not be used on entire filesystems. It works on directories trees as well.

X11, SVGA and display security

X11

It’s important for you to secure your graphical display to prevent attackers from grabbing your passwords as you type them, reading documents or information you are reading on your screen, or even using a hole to gain root access. Running remote X applications over a network also can be fraught with peril, allowing sniffers to see all your interaction with the remote system.

X has a number of access-control mechanisms. The simplest of them is host-based: you use xhost to specify what hosts are allowed access to your display. This is not very secure at all, because if someone has access to your machine, they can xhost + their machine and get in easily. Also, if you have to allow access from an untrusted machine, anyone there can compromise your display.

When using xdm (X Display Manager) to log in, you get a much better access method: MIT-MAGIC-COOKIE-1. A 128-bit "cookie" is generated and stored in your .Xauthority file. If you need to allow a remote machine access to your

display, you can use the xauth command and the information in your .Xauthority file to provide access to only that connection. See the Remote-X-Apps mini-howto, available at http://metalab.unc.edu/LDP/HOWTO/mini/Remote-X-Apps.html.

You can also use ssh (see ‘‘”, above) to allow secure X connections. This has the advantage of also being transparent to the end user, and means that no unencrypted data flows across the network.

Take a look at the Xsecurity man page for more information on X security. The safe bet is to use xdm to login to your console and then use ssh to go to remote sites on which you with to run X programs.

SVGA

SVGAlib programs are typically SUID-root in order to access all your Linux machine’s video hardware. This makes them very dangerous. If they crash, you typically need to reboot your machine to get a usable console back. Make sure any SVGA programs you are running are authentic, and can at least be somewhat trusted. Even better, don’t run them at all.

GGI (Generic Graphics Interface project)

The Linux GGI project is trying to solve several of the problems with video interfaces on Linux. GGI will move a small piece of the video code into the Linux kernel, and then control access to the video system. This means GGI will be able to restore your console at any time to a known good state. They will also allow a secure attention key, so you can be sure that there is no Trojan horse login program running on your console.
http://synergy.caltech.edu/~ggi/

Kernel Security

This is a description of the kernel configuration options that relate to security, and an explanation of what they do, and how to use them.

As the kernel controls your computer’s networking, it is important that it be very secure, and not be compromised. To prevent some of the latest networking attacks, you should try to keep your kernel version current. You can find new kernels at <ftp://ftp.kernel.org> or from your distribution vendor.

There is also a international group providing a single unified crypto patch to the mainstream linux kernel. This patch provides support for a number of cyrptographic subsystems and things that cannot be included in the mainstream kernel due to export restrictions. For more information, visit their web page at: http://www.kerneli.org

2.0 Kernel Compile Options

For 2.0.x kernels, the following options apply. You should see these options during the kernel configuration process. Many of the comments here are from ./linux/Documentation/Configure.help, which is the same document that is referenced while using the Help facility during the make config stage of compiling the kernel.

- Network Firewalls (CONFIG_FIREWALL) This option should be on if you intend to run any firewalling or masquerading on your linux machine. If it's just going to be a regular client machine, it's safe to say no.
- IP: forwarding/gatewaying (CONFIG_IP_FORWARD) If you enable IP forwarding, your Linux box essentially becomes a router. If your machine is on a network, you could be forwarding data from one network to another, and perhaps subverting a firewall that was put there to prevent this from happening. Normal dial-up users will want to disable this, and other users should concentrate on the security implications of doing this. Firewall machines will want this enabled, and used in conjunction with firewall software.
  You can enable IP forwarding dynamically using the following command:

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

  and disable it with the command:

  ```
  root#  echo 0 > /proc/sys/net/ipv4/ip_forward
  ```

  Keep in mind the files, and their sizes, do not reflect their actual sizes, and despite being zero-length, may or may not be.
- IP: syn cookies (CONFIG_SYN_COOKIES) a "SYN Attack" is a denial of service (DoS) attack that consumes all the resources on your machine, forcing you to reboot. We can't think of a reason you wouldn't normally enable this. In the 2.1 kernel series this config option mearly allows syn cookies, but does not enable them. To enable them, you have to do:
  ```
  root# echo 1 > /proc/sys/net/ipv4/tcp_syncookies
  ```
- IP: Firewalling (CONFIG_IP_FIREWALL) This option is necessary if you are going to configure your machine as a firewall, do masquerading, or wish to protect your dial-up workstation from someone entering via your PPP dial-up interface.
- IP: firewall packet logging (CONFIG_IP_FIREWALL_VERBOSE) This option gives you information about packets your firewall received, like sender, recipient, port, etc.
- IP: Drop source routed frames (CONFIG_IP_NOSR) This option should be enabled. Source routed frames contain the entire path to their destination inside of the packet. This means that routers through which the packet goes do not need to inspect it, and just forward it on. This could lead to data entering your system that may be a potential exploit.
- IP: masquerading (CONFIG_IP_MASQUERADE) If one of the computers on your local network for which your Linux box acts as a firewall wants to send something to the outside, your box can "masquerade" as that host, i.e., it forwards the traffice to the intended destination, but makes it look like it came from the firewall box itself. See http://www.indyramp.com/masq for more information.
- IP: ICMP masquerading (CONFIG_IP_MASQUERADE_ICMP) This option adds ICMP masquerading to the previous option of only masquerading TCP or UDP traffic.
- IP: transparent proxy support (CONFIG_IP_TRANSPARENT_PROXY) This enables your Linux firewall to transparently redirect any network

traffice originating from the local network and destined for a remote host to a local server, called a "transparent proxy server". This makes the local computers think they are talking to the remote end, while in fact they are connected to the local proxy. See the IP-Masquerading HOWTO and http://www.indyramp.com/masq for more information.

- IP: always defragment (CONFIG_IP_ALWAYS_DEFRAG) Generally this option is disabled, but if you are building a firewall or a masquerading host, you will want to enable it. When data is sent from one host to another, it does not always get sent as a single packet of data, but rather it is fragmented into several pieces. The problem with this is that the port numbers are only stored in the first fragment. This means that someone can insert information into the remaining packets that isn't supposed to be there. It could also prevent a teardrop attack against an internal host that is not yet itself patched against it.
- Packet Signatures (CONFIG_NCPFS_PACKET_SIGNING) This is an option that is available in the 2.1 kernel series that will sign NCP packets for stronger security. Normally you can leave it off, but it is there if you do need it.
- IP: Firewall packet netlink device (CONFIG_IP_FIREWALL_NETLINK) This is a really neat option that allows you to analyze the first 128 bytes of the packets in a user-space program, to determine if you would like to accept or deny the packet, based on its validity.

2.2 Kernel Compile Options

For 2.2.x kernels, many of the options are the same, but a few new ones have been developed. Many of the comments here are from ./linux/Documentation/Configure.help, which is the same document that is referenced while using the Help facility during the make config stage of compiling the kernel. Only the newly- added options are listed below. Consult the 2.0 description for a list of other necessary options. The most signficant change in the 2.2 kernel series is the IP firewalling code. The ipchains program is now used to install IP firewalling, instead of the ipfwadm program used in the 2.0 kernel.

- Socket Filtering (CONFIG_FILTER) For most people, it's safe to say no to this option. This option allows you to connect a userspace filter to any socket and determine if packets should be allowed or denied. Unless you have a very specific need and are capable of programming such a filter, you should say no. Also note that as of this writing, all protocols were supported except TCP.
- Port Forwarding Port Forwarding is an addition to IP Masquerading which allows some forwarding of packets from outside to inside a firewall on given ports. This could be useful if, for example, you want to run a web server behind the firewall or masquerading host and that web server should be accessible from the outside world. An external client sends a request to port 80 of the firewall, the firewall forwards this request to the web server, the web server handles the request and the results are sent through the firewall to the original client. The client thinks that the firewall machine itself is running the web server. This can also be used for load balancing if you have a farm of identical web servers behind the firewall.

Information about this feature is available from http://www.monmouth.demon.co.uk/ipsubs/portforwarding.html (to browse the WWW, you need to have access to a machine on the Internet that has a program like lynx or netscape). For general info, please see ftp://ftp.compsoc.net/users/steve/ipportfw/linux21/

- Socket Filtering (CONFIG_FILTER) Using this option, user-space programs can attach a filter to any socket and thereby tell the kernel that it should allow or disallow certain types of data to get through the socket. Linux socket filtering works on all socket types except TCP for now. See the text file ./linux/Documentation/networking/filter.txt for more information.
- IP: Masquerading The 2.2 kernel masquerading has been improved. It provides additional support for masquerading special protocols, etc. Be sure to read the IP Chains HOWTO for more information.

Kernel Devices

There are a few block and character devices available on Linux that will also help you with security.

The two devices /dev/random and /dev/urandom are provided by the kernel to provide random data at any time.

Both /dev/random and /dev/urandom should be secure enough to use in generating PGP keys, ssh challenges, and other applications where secure random numbers are requisite. Attackers should be unable to predict the next number given any initial sequence of numbers from these sources. There has been a lot of effort put in to ensuring that the numbers you get from these sources are random in every sense of the word.

The only difference is that /dev/random runs out of random bytes and it makes you wait for more to be accumulated. Note that on some systems, it can block for a long time waiting for new user-generated entry to be entered into the system. So you have to use care before using /dev/random. (Perhaps the best thing to do is to use it when you're generating sensitive keying information, and you tell the user to pound on the keyboard repeatedly until you print out "OK, enough".)

/dev/random is high quality entropy, generated from measuring the inter-interrupt times etc. It blocks until enough bits of random data are available.

/dev/urandom is similar, but when the store of entropy is running low, it'll return a cryptographically strong hash of what there is. This isn't as secure, but it's enough for most applications. You might read from the devices using something like:

```
root#  head -c 6 /dev/urandom | mmencode
```

This will print six random characters on the console, suitable for password generation. You can find mmencode in the metamail package.

See /usr/src/linux/drivers/char/random.c for a description of the algorithm.

Thanks to Theodore Y. Ts'o, Jon Lewis, and others from Linux-kernel for helping me (Dave) with this.

Network Security

Network security is becoming more and more important as people spend more and more time connected. Compromising network security is often much easier than compromising physical or local, and is much more common.

There are a number of good tools to assist with network security, and more and more of them are shipping with Linux distributions.

Packet Sniffers

One of the most common ways intruders gain access to more systems on your network is by employing a packet sniffer on a already compromised host. This "sniffer" just listens on the Ethernet port for things like passwd and login and su in the packet stream and then logs the traffic after that. This way, attackers gain passwords for systems they are not even attempting to break into. Clear-text passwords are very vulnerable to this attack.

Example: Host A has been compromised. Attacker installs a sniffer. Sniffer picks up admin logging into Host B from Host C. It gets the admin's personal password as they login to B. Then, the admin does a su to fix a problem. They now have the root password for Host B. Later the admin lets someone telnet from his account to Host Z on another site. Now the attacker has a password/login on Host Z.

In this day and age, the attacker doesn't even need to compromise a system to do this: they could also bring a laptop or pc into a building and tap into your net.

Using ssh or other encrypted password methods thwarts this attack. Things like APOP for POP accounts also prevents this attack. (Normal POP logins are very vulnerable to this, as is anything that sends clear-text passwords over the network.)

System services and tcp_wrappers

Before you put your Linux system on ANY network the first thing to look at is what services you need to offer. Services that you do not need to offer should be disabled so that you have one less thing to worry about and attackers have one less place to look for a hole.

There are a number of ways to disable services under Linux. You can look at your /etc/inetd.conf file and see what services are being offered by your inetd. Disable any that you do not need by commenting them out (# at the beginning of the line), and then sending your inetd process a SIGHUP.

You can also remove (or comment out) services in your /etc/services file. This will mean that local clients will also be unable to find the service (i.e., if you remove ftp, and try and ftp to a remote site from that machine it will fail with an "unknown service" message). It's usually not worth the trouble to remove services, since it provides no additional security. If a local person wanted to use ftp even though you had commented it out, they would make their own client that use the common FTP port and would still work fine.

Some of the services you might want to leave enabled are:

- ftp

- telnet (or ssh)
- mail, such as pop-3 or imap
- identd

If you know you are not going to use some particular package, you can also delete it entirely. rpm -e packagename under the Red Hat distribution will erase an entire package. Under debian dpkg --remove does the same thing.

Additionally, you really want to disable the rsh/rlogin/rcp utilities, including login (used by rlogin), shell (used by rcp), and exec (used by rsh) from being started in /etc/inetd.conf. These protocols are extremely insecure and have been the cause of exploits in the past.

You should check your /etc/rc.d/rcN.d, (where N is your systems run level) and see if any of the servers started in that directory are not needed. The files in /etc/rc.d/rcN.d are actually symbolic links to the directory /etc/rc.d/init.d. Renaming the files in the init.d directory has the effect of disabling all the symbolic links in /etc/rc.d/rcN.d. If you only wish to disable a service for a particular run level, rename the appropriate file by replacing the upper-case S with a lower-case s, like this:

```
root#  cd /etc/rc6.d
root#  mv S45dhcpd s45dhcpd
```

If you have BSD style rc files, you will want to check /etc/rc* for programs you don't need.

Most Linux distributions ship with tcp_wrappers "wrapping" all your TCP services. A tcp_wrapper (tcpd) is invoked from inetd instead of the real server. tcpd then checks the host that is requesting the service, and either executes the real server, or denies access from that host. tcpd allows you to restrict access to your TCP services. You should make a /etc/hosts.allow and add in only those hosts that need to have access to your machine's services.

If you are a home dialup user, we suggest you deny ALL. tcpd also logs failed attempts to access services, so this can give alert you if you are under attack. If you add new services, you should be sure to configure them to use tcp_wrappers if they are TCP based. For example, a normal dial-up user can prevent outsiders from connecting to his machine, yet still have the ability to retrieve mail, and make network connections to the Internet. To do this, you might add the following to your /etc/hosts.allow:

```
ALL: 127.
```

And of course /etc/hosts.deny would contain:

```
ALL: ALL
```

which will prevent external connections to your machine, yet still allow you from the inside to connect to servers on the Internet.

Keep in mind that tcp_wrappers only protect services executed from inetd, and a select few others. There very well may be other services running on your machine. You can use netstat -ta to find a list of all the services your machine is offering.

Verify Your DNS Information

Keeping up-to-date DNS information about all hosts on your network can help to increase security. If an unauthorized host becomes connected to your

network, you can recognize it by its lack of a DNS entry. Many services can be configured to not accept connections from hosts that do not have valid DNS entries.

identd

identd is a small program that typically runs out of your inetd server. It keeps track of what user is running what TCP service, and then reports this to whoever requests it.

Many people misunderstand the usefulness of identd, and so disable it or block all off site requests for it. identd is not there to help out remote sites. There is no way of knowing if the data you get from the remote identd is correct or not. There is no authentication in identd requests.

Why would you want to run it then? Because it helps you out, and is another data-point in tracking. If your identd is un compromised, then you know it's telling remote sites the user-name or uid of people using TCP services. If the admin at a remote site comes back to you and tells you user so-and-so was trying to hack into their site, you can easily take action against that user. If you are not running identd, you will have to look at lots and lots of logs, figure out who was on at the time, and in general take a lot more time to track down the user.

The identd that ships with most distributions is more configurable than many people think. You can disable it for specific users (they can make a .noident file), you can log all identd requests (We recommend it), you can even have identd return a uid instead of a user name or even NO-USER.

SATAN, ISS, and Other Network Scanners

There are a number of different software packages out there that do port and service based scanning of machines or networks. SATAN, ISS, SAINT, and Nessus are some of the more well-known ones. This software connects to the target machine (or all the target machines on a network) on all the ports they can, and try to determine what service is running there. Based on this information, you can tell if the machine is vulnerable to a specific exploit on that server.

SATAN (Security Administrator's Tool for Analyzing Networks) is a port scanner with a web interface. It can be configured to do light, medium, or strong checks on a machine or a network of machines. It's a good idea to get SATAN and scan your machine or network, and fix the problems it finds. Make sure you get the copy of SATAN from metalab <http://metalab.unc.edu/pub/packages/security/Satan-for-Linux/> or a reputable FTP or web site. There was a Trojan copy of SATAN that was distributed out on the net. http://www.trouble.org/~zen/satan/satan.html. Note that SATAN has not been updated in quite a while, and some of the other tools below might do a better job.

ISS (Internet Security Scanner) is another port-based scanner. It is faster than Satan, and thus might be better for large networks. However, SATAN tends to provide more information.

Abacus is a suite of tools to provide host based security and intrusion detection. look at it's home page on the web for more information. http://www.psionic.com/abacus/

SAINT is a updated version of SATAN. It is web based and has many more up to date tests than SATAN. You can find out more about it at: http://www.wwdsi.com/~saint

Nessus is a free security scanner. It has a GTK graphical interface for ease of use. It is also designed with a very nice plugin setup for new port scanning tests. For more information, take a look at: http://www.nessus.org

Detecting Port Scans

There are some tools designed to alert you to probes by SATAN and ISS and other scanning software. However, liberal use of tcp_wrappers, and make sure to look over your log files regularly, you should be able to notice such probes. Even on the lowest setting, SATAN still leaves traces in the logs on a stock Red Hat system.

There are also "stealth" port scanners. A packet with the TCP ACK bit set (as is done with established connections) will likely get through a packet-filtering firewall. The returned RST packet from a port that _had no established session_ can be taken as proof of life on that port. I don't think TCP wrappers will detect this.

sendmail , qmail and MTA's

One of the most important services you can provide is a mail server. Unfortunately, it is also one of the most vulnerable to attack, simply due to the number of tasks it must perform and the privileges it typically needs.

If you are using sendmail it is very important to keep up on current versions. sendmail has a long long history of security exploits. Always make sure you are running the most recent version from http://www.sendmail.org.

Keep in mind that sendmail does not have to be running in order for you to send mail. If you are a home user, you can disable sendmail entirely, and simply use your mail client to send mail. You might also choose to remove the "-bd" flag from the sendmail startup file, thereby disabling incoming requests for mail. In other words, you can execute sendmail from your startup script using the following instead:

```
# /usr/lib/sendmail -q15m
```

This will cause sendmail to flush the mail queue every fifteen minutes for any messages that could not be successfully delivered on the first attempt.

Many administrators choose not to use sendmail, and instead choose one of the other mail transport agents. You might consider switching over to qmail. qmail was designed with security in mind from the ground up. It's fast, stable, and secure. Qmail can be found at http://www.qmail.org

In direct competition to qmail is "postfix", written by Wietse Venema, the author of tcp_wrappers and other security tools. Formerly called vmailer, and sponsored by IBM, this is also a mail transport agent written from the ground up with security in mind. You can find more information about vmailer at http://www.postfix.org

Denial of Service Attacks

A "Denial of Service" (DoS) attack is one where the attacker tries to make some resource too busy to answer legitimate requests, or to deny legitimate users access to your machine.

Denial of service attacks have increased greatly in recent years. Some of the more popular and recent ones are listed below. Note that new ones show up all the time, so this is just a few examples. Read the Linux security lists and the bugtraq list and archives for more current information.

- SYN Flooding - SYN flooding is a network denial of service attack. It takes advantage of a "loophole" in the way TCP connections are created. The newer Linux kernels (2.0.30 and up) have several configurable options to prevent SYN flood attacks from denying people access to your machine or services. See ‘‘Kernel Security” for proper kernel protection options.
- Pentium "F00F" Bug - It was recently discovered that a series of assembly codes sent to a genuine Intel Pentium processor would reboot the machine. This affects every machine with a Pentium processor (not clones, not Pentium Pro or PII), no matter what operating system it’s running. Linux kernels 2.0.32 and up contain a work around for this bug, preventing it from locking your machine. Kernel 2.0.33 has an improved version of the kernel fix, and is suggested over 2.0.32. If you are running on a Pentium, you should upgrade now!
- Ping Flooding - Ping flooding is a simple brute-force denial of service attack. The attacker sends a "flood" of ICMP packets to your machine. If they are doing this from a host with better bandwidth than yours, your machine will be unable to send anything on the network. A variation on this attack, called "smurfing", sends ICMP packets to a host with your machine’s return IP, allowing them to flood you less detectably. You can find more information about the "smurf" attack at http://www.quadrunner.com/~chuegen/smurf.txt If you are ever under a ping flood attack, use a tool like tcpdump to determine where the packets are coming from (or appear to be coming from), then contact your provider with this information. Ping floods can most easily be stopped at the router level or by using a firewall.
- Ping o’Death - The Ping o’Death attack sends ICMP ECHO REQUEST packets that are too large to fit in the kernel data structures intended to store them. Because sending a single, large (65,510 bytes) "ping" packet to many systems will cause them to hang or even crash, this problem was quickly dubbed the "Ping o’Death." This one has long been fixed, and is no longer anything to worry about.
- Teardrop / New Tear - One of the most recent exploits involves a bug present in the IP fragmentation code on Linux and Windows platforms. It is fixed in kernel version 2.0.33, and does not require selecting any kernel compile-time options to utilize the fix. Linux is apparently not vulnerable to the "newtear" exploit.

You can find code for most exploits, and a more in-depth description of how they work, at http://www.rootshell.com using their search engine.

NFS (Network File System) Security.

NFS is a very widely-used file sharing protocol. It allows servers running nfsd and mountd to "export" entire filesystems to other machines using NFS filesystem support built in to their kernels (or some other client support if they are not Linux machines). mountd keeps track of mounted filesystems in /etc/mtab, and can display them with showmount.

Many sites use NFS to serve home directories to users, so that no matter what machine in the cluster they login to, they will have all their home files.

There is some small amount of security allowed in exporting filesystems. You can make your nfsd map the remote root user (uid=0) to the nobody user, denying them total access to the files exported. However, since individual users have access to their own (or at least the same uid) files, the remote root user can login or su to their account and have total access to their files. This is only a small hindrance to an attacker that has access to mount your remote filesystems.

If you must use NFS, make sure you export to only those machines that you really need to. Never export your entire root directory; export only directories you need to export.

See the NFS HOWTO for more information on NFS, available at http://metalab.unc.edu/mdw/HOWTO/NFS-HOWTO.html

NIS (Network Information Service) (formerly YP).

Network Information service (formerly YP) is a means of distributing information to a group of machines. The NIS master holds the information tables and converts them into NIS map files. These maps are then served over the network, allowing NIS client machines to get login, password, home directory and shell information (all the information in a standard /etc/passwd file). This allows users to change their password once and have it take effect on all the machines in the NIS domain.

NIS is not at all secure. It was never meant to be. It was meant to be handy and useful. Anyone that can guess the name of your NIS domain (anywhere on the net) can get a copy of your passwd file, and use "crack" and "John the Ripper" against your users' passwords. Also, it is possible to spoof NIS and do all sorts of nasty tricks. If you must use NIS, make sure you are aware of the dangers.

There is a much more secure replacement for NIS, called NIS+. Check out the NIS HOWTO for more information:
http://metalab.unc.edu/mdw/HOWTO/NIS-HOWTO.html

Firewalls

Firewalls are a means of controlling what information is allowed into and out of your local network. Typically the firewall host is connected to the Internet and your local LAN, and the only access from your LAN to the Internet is through the firewall. This way the firewall can control what passes back and forth from the Internet and your lan.

There are a number of types of firewalls and methods of setting them up. Linux machines make pretty good firewalls. Firewall code can be built right into 2.0 and higher kernels. The ipfwadm for 2.0 kernels, or ipchains for 2.2

kernels, user-space tools allows you to change, on the fly, the types of network traffic you allow. You can also log particular types of network traffic.

Firewalls are a very useful and important technique in securing your network. However, never think that because you have a firewall, you don't need to secure the machines behind it. This is a fatal mistake. Check out the very good Firewall-HOWTO at your latest metalab archive for more information on firewalls and Linux. http://metalab.unc.edu/mdw/HOWTO/Firewall-HOWTO.html

More information can also be found in the IP-Masquerade mini-howto: http://metalab.unc.edu/mdw/HOWTO/mini/IP-Masquerade.html

More information on ipfwadm (The tool that lets you change settings on your firewall, can be found at it's home page: http://www.xos.nl/linux/ipfwadm/

If you have no experience with firewalls, and plan to set up one for more than just a simple security policy, the Firewalls book by O'Reilly and Associates or other online firewall document is mandatory reading. Check out http://www.ora.com for more information. The National Institute of Standards and Technology have put together an excellent document on firewalls. Although dated 1995, it is still quite good. You can find it at http://csrc.nist.gov/nistpubs/800-10/main.html. Also of interest includes:

- The Freefire Project -- a list of freely-available firewall tools, available at http://sites.inka.de/sites/lina/freefire-l/index_en.html
- SunWorld Firewall Design -- written by the authors of the O'Reilly book, this provides a rough introduction to the different firewall types. It's available at http://www.sunworld.com/swol-01-1996/swol-01-firewall.html

IP Chains - Linux Kernel 2.2.x Firewalling

Linux IP Firewalling Chains is an update to the 2.0 Linux firewalling code for the 2.2 kernel. It has a great deal more features than previous implementations, including:

- More flexible packet manipulations
- More complex accounting
- Simple policy changes possible atomically
- Fragments can be explicitly blocked, denied, etc.
- Logs suspicious packets.
- Can handle protocols other than ICMP/TCP/UDP.

If you are currently using ipfwadm on your 2.0 kernel, there are scripts available to convert the ipfwadm command format to the format ipchains uses.

Be sure to read the IP Chains HOWTO for further information. It is avilable at http://www.rustcorp.com/linux/ipchains/HOWTO.html

VPN's - Virtual Private Networks

VPN's are a way to establish a "virtual" network on top of some already existing network. This virtual network often is encrypted and passes traffic only to and from some known entities that have joined the network. VPN's are

often used to connect someone working at home over the public internet to a internal company network by using a encrypted virtual network.

If you are running a linux masquerading firewall and need to pass MS PPTP (Microsoft's VPN point to point product) packets, there is a linux kernel patch out to do just that. See: ip-masq-vpn.

There are several linux VPN solutions available:

- vpnd. See the http://www.crosswinds.net/nuremberg/~anstein/unix/vpnd.html.
- Free S/Wan, available at http://www.xs4all.nl/~freeswan/
- ssh can be used to construct a VPN. See the VPN mini-howto for more information.
- vps (virtual private server) at http://www.strongcrypto.com.

See also the section on IPSEC for pointers and more information.

Security Preparation (before you go on-line)

Ok, so you have checked over your system, and determined it's as secure as feasible, and you're ready to put it online. There are a few things you should now do in order to prepare for an intrusion, so you can quickly disable the intruder, and get back up and running.

Make a Full Backup of Your Machine

Discussion of backup methods and storage is beyond the scope of this document, but here are a few words relating to backups and security:

If you have less than 650mb of data to store on a partition, a CD-R copy of your data is a good way to go (as it's hard to tamper with later, and if stored properly can last a long time). Tapes and other re-writable media should be write-protected as soon as your backup is complete, and then verified to prevent tampering. Make sure you store your backups in a secure off-line area. A good backup will ensure that you have a known good point to restore your system from.

Choosing a Good Backup Schedule

A six-tape cycle is easy to maintain. This includes four tapes for during the week, one tape for even Fridays, and one tape for odd Fridays. Perform an incremental backup every day, and a full backup on the appropriate Friday tape. If you make some particularly important changes or add some important data to your system, a full backup might well be in order.

Backup Your RPM or Debian File Database

In the event of an intrusion, you can use your RPM database like you would use tripwire, but only if you can be sure it too hasn't been modified. You should copy the RPM database to a floppy, and keep this copy off-line at all times. The Debian distribution likely has something similar.

The files /var/lib/rpm/fileindex.rpm and /var/lib/rpm/packages.rpm most likely won't fit on a single floppy. But if Compressed, each should fit on a seperate floppy.

Now, when your system is compromised, you can use the command:

```
root#  rpm -Va
```

to verify each file on the system. See the rpm man page, as there are a few other options that can be included to make it less verbose. Keep in mind you must also be sure your RPM binary has not been compromised.

This means that every time a new RPM is added to the system, the RPM database will need to be rearchived. You will have to decide the advantages versus drawbacks.

Keep Track of Your System Accounting Data

It is very important that the information that comes from syslog has not been compromised. Making the files in /var/log readable and writable by only a limited number of users is a good start.

Be sure to keep an eye on what gets written there, especially under the auth facility. Multiple login failures, for example, can indicate an attempted break-in.

Where to look for your log file will depend on your distribution. In a Linux system that conforms to the "Linux Filesystem Standard", such as Red Hat, you will want to look in /var/log and check messages, mail.log, and others.

You can find out where your distribution is logging to by looking at your /etc/syslog.conf file. This is the file that tells syslogd (the system logging daemon) where to log various messages.

You might also want to configure your log-rotating script or daemon to keep logs around longer so you have time to examine them. Take a look at the logrotate package on recent Red Hat distributions. Other distributions likely have a similar process.

If your log files have been tampered with, see if you can determine when the tampering started, and what sort of things appeared to be tampered with. Are there large periods of time that cannot be accounted for? Checking backup tapes (if you have any) for untampered log files is a good idea.

Log files are typically modified by the intruder in order to cover his tracks, but they should still be checked for strange happenings. You may notice the intruder attempting to gain entrance, or exploit a program in order to obtain the root account. You might see log entries before the intruder has time to modify them.

You should also be sure to seperate the auth facility from other log data, including attempts to switch users using su, login attempts, and other user accounting information.

If possible, configure syslog to send a copy of the most important data to a secure system. This will prevent an intruder from covering his tracks by deleting his login/su/ftp/etc attempts. See the syslog.conf man page, and refer to the @ option.

There are several more advanced syslogd programs out there. Take a look at http://www.core-sdi.com/ssyslog/ for Secure Syslog. Secure Syslog allows you to encrypt your syslog entries and make sure no one has tampered with them.

Another syslogd with more features is syslog-ng. It allows you a lot more flexability in your logging and also can has your remote syslog streams to prevent tampering.

Finally, log files are much less useful when no one is reading them. Take some time out every once in a while to look over your log files, and get a feeling for what they look like on a normal day. Knowing this can help make unusual things stand out.

Apply All New System Updates.

Most Linux users install from a CD-ROM. Due to the fast-paced nature of security fixes, new (fixed) programs are always being released. Before you connect your machine to the network, it's a good idea to check with your distribution's ftp site and get all the updated packages since you received your distribution CD-ROM. Many times these packages contain important security fixes, so it's a good idea to get them installed.

What To Do During and After a Breakin

So you have followed some of the advice here (or elsewhere) and have detected a break-in? The first thing to do is to remain calm. Hasty actions can cause more harm than the attacker would have.

Security Compromise Underway.

Spotting a security compromise under way can be a tense undertaking. How you react can have large consequences.

If the compromise you are seeing is a physical one, odds are you have spotted someone who has broken into your home, office or lab. You should notify your local authorities. In a lab, you might have spotted someone trying to open a case or reboot a machine. Depending on your authority and procedures, you might ask them to stop, or contact your local security people.

If you have detected a local user trying to compromise your security, the first thing to do is confirm they are in fact who you think they are. Check the site they are logging in from. Is it the site they normally log in from? No? Then use a non-electronic means of getting in touch. For instance, call them on the phone or walk over to their office/house and talk to them. If they agree that they are on, you can ask them to explain what they were doing or tell them to cease doing it. If they are not on, and have no idea what you are talking about, odds are this incident requires further investigation. Look into such incidents , and have lots of information before making any accusations.

If you have detected a network compromise, the first thing to do (if you are able) is to disconnect your network. If they are connected via modem, unplug the modem cable; if they are connected via ethernet, unplug the Ethernet cable. This will prevent them from doing any further damage, and they will probably see it as a network problem rather than detection.

If you are unable to disconnect the network (if you have a busy site, or you do not have physical control of your machines), the next best step is to use something like tcp_wrappers or ipfwadm to deny access from the intruder's site.

If you can't deny all people from the same site as the intruder, locking the user's account will have to do. Note that locking an account is not an easy thing. You have to keep in mind .rhosts files, FTP access, and a host of possible backdoors).

After you have done one of the above (disconnected the network, denied access from their site, and/or disabled their account), you need to kill all their user processes and log them off.

You should monitor your site well for the next few minutes, as the attacker will try to get back in. Perhaps using a different account, and/or from a different network address.

Security Compromise has already happened

So you have either detected a compromise that has already happened or you have detected it and locked (hopefully) the offending attacker out of your system. Now what?

Closing the Hole

If you are able to determine what means the attacker used to get into your system, you should try to close that hole. For instance, perhaps you see several FTP entries just before the user logged in. Disable the FTP service and check and see if there is an updated version, or if any of the lists know of a fix.

Check all your log files, and make a visit to your security lists and pages and see if there are any new common exploits you can fix. You can find Caldera security fixes at http://www.caldera.com/tech-ref/security/. Red Hat has not yet seperated their security fixes from bug fixes, but their distribution errata is available at http://www.redhat.com/errata

Debian now has a security mailing list and web page. See: http://www.debian.com/security/ for more information.

It is very likely that if one vendor has released a security update, that most other Linux vendors will as well.

There is now a linux security auditing project. They are methodically going through all the user space utilities and looking for possible security exploits and overflows. From their announcement:

> "We are attempting a systematic audit of Linux sources with a view to being as secure as OpenBSD. We have already uncovered (and fixed) some problems, but more help is welcome. The list is unmoderated and also a useful resource for general security discussions. The list address is: security-audit@ferret.lmh.ox.ac.uk To subscribe, send a mail to: security-audit-subscribe@ferret.lmh.ox.ac.uk"

If you don't lock the attacker out, they will likely be back. Not just back on your machine, but back somewhere on your network. If they were running a packet sniffer, odds are good they have access to other local machines.

Assessing the Damage

The first thing is to assess the damage. What has been compromised? If you are running an Integrity Checker like Tripwire, you can use it to perform an integrity check, and should help to tell you. If not, you will have to look around at all your important data.

Since Linux systems are getting easier and easier to install, you might consider saving your config files and then wiping your disk(s) and reinstalling, then restoring your user files from backups and your config files. This will ensure that you have a new, clean system. If you have to backup files from the compromised system, be especially cautious of any binaries that you restore, as they may be Trojan horses placed there by the intruder.

Re-installation should be considered mandatory upon an intruder obtaining root access. Additionally, you'd like to keep any evidence there is, so having a spare disk in the safe may make sense.

Then you have to worry about how long ago the compromise happened, and whether the backups hold any damaged work. More on backups later.

Backups, Backups, Backups!

Having regular backups is a godsend for security matters. If your system is compromised, you can restore the data you need from backups. Of course, some data is valuable to the attacker too, and they will not only destroy it, they will steal it and have their own copies; but at least you will still have the data.

You should check several backups back into the past before restoring a file that has been tampered with. The intruder could have compromised your files long ago, and you could have made many successful backups of the compromised file!!!

Of course, there are also a raft of security concerns with backups. Make sure you are storing them in a secure place. Know who has access to them. (If an attacker can get your backups, they can have access to all your data without you ever knowing it.)

Tracking Down the Intruder.

Ok, you have locked the intruder out, and recovered your system, but you're not quite done yet. While it is unlikely that most intruders will ever be caught, you should report the attack.

You should report the attack to the admin contact at the site where the attacker attacked your system. You can look up this contact with whois or the Internic database. You might send them an email with all applicable log entries and dates and times. If you spotted anything else distinctive about your intruder, you might mention that too. After sending the email, you should (if you are so inclined) follow up with a phone call. If that admin in turn spots your attacker, they might be able to talk to the admin of the site where they are coming from and so on.

Good crackers often use many intermediate systems, some (or many) of which may not even know they have been compromised. Trying to track a cracker back to their home system can be difficult. Being polite to the admins you talk to can go a long way to getting help from them. You should also notify any

security organizations you are a part of (CERT <http://www.cert.org/> or similar), as well as your Linux system vendor.

Security Sources

There are a LOT of good sites out there for Unix security in general and Linux security specifically. It's very important to subscribe to one (or more) of the security mailing lists and keep current on security fixes. Most of these lists are very low volume, and very informative.

FTP Sites

CERT is the Computer Emergency Response Team. They often send out alerts of current attacks and fixes. See ftp://ftp.cert.org for more information.

Replay (http://www.replay.com) has archives of many security programs. Since they are outside the US, they don't need to obey US crypto restrictions.

Matt Blaze is the author of CFS and a great security advocate. Matt's archive is available at ftp://ftp.research.att.com/pub/mab <ftp://ftp.research.att.com/pub/mab>

tue.nl is a great security FTP site in the Netherlands. ftp.win.tue.nl

Web Sites

- The Hacker FAQ is a FAQ about hackers: http://www.tuxedo.org/~esr/faqs/hacker-howto.html
- The COAST archive has a large number of Unix security programs and information: http://www.cerias.purdue.edu/coast/archive/index.html
- SuSe Security Page: http://www.suse.de/security/
- Rootshell.com is a great site for seeing what exploits are currently being used by crackers: http://www.rootshell.com/
- BUGTRAQ puts out advisories on security issues: http://www.securityfocus.com/forums/bugtraq/intro.html
- CERT, the Computer Emergency Response Team, puts out advisories on common attacks on unix platforms: http://www.cert.org/
- Dan Farmer is the author of SATAN and many other security tools. His home site has some interesting security survey information, as well as security tools: http://www.trouble.org
- The Linux security WWW is a good site for Linux security information: http://www.ecst.csuchico.edu/~jtmurphy/
- Infilsec has a vulnerability engine that can tell you what vunerabilities affect a specific platform: http://www.infilsec.com/vulnerabilities/
- CIAC sends out periodic security bulletins on common exploits: http://ciac.llnl.gov/cgi-bin/index/bulletins
- A good starting point for Linux Pluggable Authentication modules can be found at http://www.kernel.org/pub/linux/libs/pam/. The debian project

has a web page for their security fixes and information. It is at http://www.debian.com/security/.

- WWW Security FAQ, written by Lincoln Stein, is a great web security reference. Find it at http://www.w3.org/Security/Faq/www-security-faq.html

Mailing Lists

Bugtraq: To subscribe to bugtraq, send mail to listserv@netspace.org containing the message body subscribe bugtraq. (see links above for archives).

CIAC: Send e-mail to majordomo@tholia.llnl.gov. In the BODY (not subject) of the message put (either or both): subscribe ciac-bulletin

Red Hat has a number of mailing lists, the most important of which is the redhat-announce list. You can read about security (and other) fixes as soon as they come out. Send email to majordomo@redhat.com and put subscribe redhat-announce.

The Debian project has a security mailing list that covers their security fixes. see http://www.debian.com/security/ for more information.

Books - Printed Reading Material

There are a number of good security books out there. This section lists a few of them. In addition to the security specific books, security is covered in a number of other books on system administration.

Building Internet Firewalls By D. Brent Chapman & Elizabeth D. Zwicky 1st Edition September 1995 ISBN: 1-56592-124-0

Practical UNIX & Internet Security, 2nd Edition By Simson Garfinkel & Gene Spafford, 2nd Edition April 1996 ISBN: 1-56592-148-8

Computer Security Basics By Deborah Russell & G.T. Gangemi, Sr. 1st Edition July 1991 ISBN: 0-937175-71-4

Linux Network Administrator's Guide By Olaf Kirch 1st Edition January 1995 ISBN: 1-56592-087-2

PGP: Pretty Good Privacy By Simson Garfinkel 1st Edition December 1994 ISBN: 1-56592-098-8

Computer Crime A Crimefighter's Handbook By David Icove, Karl Seger & William VonStorch (Consulting Editor Eugene H. Spafford) 1st Edition August 1995 ISBN: 1-56592-086-4

Glossary

- authentication: The property of knowing that the data received is the same as the data that was sent, and that the claimed sender is in fact the actual sender.
- bastion Host: A computer system that must be highly secured because it is vulnerable to attack, usually because it is exposed to the Internet and is a main point of contact for users of internal networks. It gets its name from the highly fortified projects on the outer walls of medieval castles. Bastions overlook critical areas of defense, usually having strong walls,

room for extra troops, and the occasional useful tub of boiling hot oil for discouraging attackers.

- buffer overflow: Common coding style is to never allocate large enough buffers, and to not check for overflows. When such buffers overflow, the executing program (daemon or set-uid program) can be tricked in doing some other things. Generally this works by overwriting a function's return address on the stack to point to another location.
- denial of service: A denial of service attack is when an attacker consumes the resources on your computer for things it was not intended to be doing, thus preventing normal use of your network resources for legimite purposes.
- dual-homed Host: A general-purpose computer system that has at least two network interfaces.
- firewall: A component or set of components that restricts access between a protected network and the Internet, or between other sets of networks.
- host: A computer system attached to a network.
- IP spoofing: IP Spoofing is a complex technical attack that is made up of several components. It is a security exploit that works by tricking computers in a trust-relationship that you are someone that you really aren't. There is an extensive paper written by daemon9, route, and infinity in the Volume Seven, Issue fourty-Eight issue of Phrack Magazine.
- non-repudiation: The property of a receiver being able to prove that the sender of some data did in fact send the data even though the sender might later deny ever having sent it.
- packet: The fundamental unit of communication on the Internet.
- packet filtering: The action a device takes to selectively control the flow of data to and from a network. Packet filters allow or block packets, usually while routing them from one network to another (most often from the Internet to an internal network, and vice-versa). To accomplish packet filtering, you set up rules that specify what types of packets (those to or from a particular IP address or port) are to be allowed and what types are to be blocked.
- perimeter network: A network added between a protected network and an external network, in order to provide an additional layer of security. A perimeter network is sometimes called a DMZ.
- proxy server: A program that deals with external servers on behalf of internal clients. Proxy clients talk to proxy servers, which relay approved client requests to real servers, and relay answers back to clients.
- superuser: An informal name for root.

Frequently Asked Questions

1. Is it more secure to compile driver support directly into the kernel, instead of making it a module?
   Answer: Some people think it is better to disable the ability to load device drivers using modules, because an intruder could load a Trojan module or a module that could affect system security.

However, in order to load modules, you must be root. The module object files are also only writable by root. This means the intruder would need root access to insert a module. If the intruder gains root access, there are more serious things to worry about than whether he will load a module.

Modules are for dynamically loading support for a particular device that may be infrequently used. On server machines, or firewalls for instance, this is very unlikely to happen. For this reason, it would make more sense to compile support directly into the kernel for machines acting as a server. Modules are also slower than support compiled directly in the kernel.

2. Why does logging in as root from a remote machine always fail?
   Answer: See ‘‘Root Security”. This is done intentionally to prevent remote users from attempting to connect via telnet to your machine as root, which is a serious security vulnerability. Don’t forget: potential intruders have time on their side, and can run automated programs to find your password.
3. How do I enable shadow passwords on my Red Hat 4.2 or 5.x Linux box?
   Answer: Shadow passwords is a mechanism for storing your password in a file other than the normal /etc/passwd file. This has several advantages. The first one is that the shadow file, /etc/shadow, is only readable by root, unlike /etc/passwd, which must remain readable by everyone. The other advantage is that as the administrator, you can enable or disable accounts without everyone knowing the status of other users’ accounts.

   The /etc/passwd file is then used to store user and group names, used by programs like /bin/ls to map the user ID to the proper username in a directory listing.

   The /etc/shadow file then only contains the username and his/her password, and perhaps accounting information, like when the account expires, etc.

   To enable shadow passwords, run pwconv as root, and /etc/shadow should now exist, and be used by applications. Since you are using RH 4.2 or above, the PAM modules will automatically adapt to the change from using normal /etc/passwd to shadow passwords without any other change.

   Since you’re interested in securing your passwords, perhaps you would also be interested in generating good passwords to begin with. For this you can use the pam_cracklib module, which is part of PAM. It runs your password against the Crack libraries to help you decide if it is too easily guessable by password cracking programs.
4. How can I enable the Apache SSL extensions?
   Answer:
   1.Get SSLeay 0.8.0 or later from ftp://ftp.psy.uq.oz.au/pub/Crypto/SSL
   2.Build and test and install it!
   3.Get Apache 1.2.5 source ***1.3.10 is about to be released***
   4.Get Apache SSLeay extensions from here
   ftp://ftp.ox.ac.uk/pub/crypto/SSL/apache_1.2.5+ssl_1.13.tar.gz
   5.Unpack it in the apache-1.2.5 source directory and patch Apache as per the README.
   6.Configure and build it.

You might also try Replay Associates which has many pre-built packages, and is located outside of the United States.

5. How can I manipulate user accounts, and still retain security?
   Answer: The Red Hat distribution, especially RH5.0, contains a great number of tools to change the properties of user accounts.

   The pwconv and unpwconv programs can be used to convert between shadow and non-shadowed passwords.

   The pwck and grpck programs can be used to verify proper organization of the passwd and group files.

   The useradd, usermod, and userdel programs can be used to add, delete and modify user accounts. The groupadd, groupmod, and groupdel programs will do the same for groups. Group passwords can be created using gpasswd.

   All these programs are "shadow-aware" -- that is, if you enable shadow they will use /etc/shadow for password information, otherwise it won't.

   See the respective man pages for further information.

6. How can I password protect specific HTML documents using Apache?
   I bet you didn't know about http://www.apacheweek.org, did you?
   You can find information on user Authentication at http://www.apacheweek.com/features/userauth as well as other web server security tips from http://www.apache.org/docs/misc/security_tips.html

Conclusion

By subscribing to the security alert mailing lists, and keeping current, you can do a lot towards securing your machine. If you pay attention to your log files and run something like tripwire regularly, you can do even more.

A reasonable level of computer security is not difficult to maintain on a home machine. More effort is required on business machines, but Linux can indeed be a secure platform. Due to the nature of Linux development, security fixes often come out much faster than they do on commercial operating systems, making Linux an ideal platform when security is a requirement.

Acknowledgements

Information here is collected from many sources. Thanks to the following that either indirectly or directly have contributed: following who either indirectly or directly have contributed:

Rob Riggs rob@DevilsThumb.com

S. Coffin scoffin@netcom.com

Viktor Przebinda viktor@CRYSTAL.MATH.ou.edu

Roelof Osinga roelof@eboa.com

Kyle Hasselbacher kyle@carefree.quux.soltc.net

David S. Jackson dsj@dsj.net

Todd G. Ruskell ruskell@boulder.nist.gov

Rogier Wolff R.E.Wolff@BitWizard.nl

Antonomasia ant@notatla.demon.co.uk

Nic Bellamy sky@wibble.net

Eric Hanchrow offby1@blarg.net

Robert J. Bergerrberger@ibd.com

Ulrich Alpers lurchi@cdrom.uni-stuttgart.de

David Noha dave@c-c-s.com

The following have translated this HOWTO into various other languages!

A special thank you to all of them for help spreading the linux word...

Polish: Ziemek Borowski ziembor@FAQ-bot.ZiemBor.Waw.PL

Japanese: FUJIWARA Teruyoshi fjwr@mtj.biglobe.ne.jp

Indonesian: Tedi Heriyanto 22941219@students.ukdw.ac.id